When owl:sameAs is the Same: Experimenting Online Resolution of Identity with SPARQL queries to Linked Open Data Sources

Raphaël Gazzotti[a], Fabien Gandon[b]
Université Côte d'Azur, Inria, CNRS, I3S, Sophia-Antipolis, France
name.surname@inria.fr

Keywords: Equivalence links, Coreference resolution, SPARQL, Linked Data, Data curation, sameAs.

Abstract: Equivalence links are the cornerstone of Linked Data and their integration. However, it is not easy to establish and manipulate them, since the Web is always evolving with datasets emerging and disappearing. Inconsistencies may also be present on the Web, leading to erroneous assertions and inferences. We propose a method to identify `owl:sameAs` relationships of a resource relying on online SPARQL querying of distributed datasets and to correct results using declarative curation rules. We also exploit and inspect the quality of `owl:InverseFunctionalProperty` and `owl:FunctionalProperty` relationships, using the definitions given by their schemata, endpoints and a voting approach. We evaluate our method on an existing benchmark and compare to state of the art baselines. We show that a heuristic approach can retrieve high quality equivalence links without requiring the extraction of all the alleged existing equivalence relations.

1 INTRODUCTION

The ability to establish links is key to weaving the Web in general and the semantic Web in particular (Gandon, 2018). But the Web is in constant evolution with resources added and deleted all the time. Linked data, in particular, rely on our ability to establish links between the different datasets on the Web and as such, the detection of equivalence links is a central task. And as the Web evolves, this linking has to evolve with it. Establishing equivalences between resources is also key to data integration use cases to join knowledge graphs with different provenances. More recently this ability to combine such graphs also proved important in machine learning approaches relying on embeddings based on a set of linked graphs to capture the semantics surrounding a concept more accurately and more richly.

Approaches discovering equivalence links that rely on a snapshot of the Web run the risk of capturing relationships that already belong to the past. Equivalence links can also be retrieved by exploiting OWL semantics, e.g., properties of type (inverse) functional can indicate the uniqueness of a resource, leading to the inference of `owl:sameAs` relationships when different URIs are used as their subject or object (Allemang et al., 2020). Moreover, data on the Web is of variable quality, which requires caution in using it. Therefore, there is a need for on-demand online search and reasoning for equivalence relations. To tackle this problem, we explored the research question: *Can valid owl:sameAs relationships for a given URI be detected automatically and online?*. In this article, we answer the following sub-questions:

- Where to find and how to retrieve SPARQL endpoint information to be explored for equivalence detection?
- How to identify and correct misinformation about `owl:sameAs` statements?
- How to detect wrong type for assserted `owl:InverseFunctionalProperty` and `owl:FunctionalProperty` properties?

The paper is structured as follows. We survey the related work in Section 2 and position our contribution. In Section 3 we introduce the definition of the equivalence detection task and the vocabulary we used to solve it. Then, in Section 4 we describe how we proceed to obtain information on datasets and their SPARQL endpoints. Section 5 details how we proceed to collect and curate equivalence links. We evaluate the quality of the retrieved equivalent links on a public benchmark, comparing to state of the art baselines, and discuss our results in Section 6. We conclude with some perspectives in Section 7.

[a] https://orcid.org/0000-0002-5618-9776
[b] https://orcid.org/0000-0003-0543-1232

2 RELATED WORK

Surprisingly, there are not so many available services to establish `owl:sameAs` relationships for a given URI.

The sameAs.org (Jaffri et al., 2008) is one of the pioneers to propose a service of URIs coresolution. The equivalences are collected through different RDF files and SPARQL endpoints chosen by the authors.

Equivalent URIs are stored with different identifiers depending on the context. Different coreference contexts are captured by different coreference resolution services, because depending on the usage and context, an equivalence either holds or not. A context is represented as a bundle attached to a URI. However the approach mixes different predicates which can be far from a `owl:sameAs`, e.g., `ov:similarTo`[1] is a property for things that are not linked by `owl:sameAs`, but that are similar to a certain extent. The system also does not ensure that resources are actually equivalent and proposes only one concept of equivalence. Therefore, it does not allow to distinguish different notions of equivalence as indicated by (Halpin et al., 2010) and defined in the Similarity Ontology.[2] In addition, with the online portal, it is unfortunately not possible to distinguish the different contexts considered for a given resource. Unlike the other works, sameAs.org does not use a curation algorithm for `owl:sameAs` relationships.

The LODsyndesis[3] (Mountantonakis and Tzitzikas, 2018) platform, in addition to providing various services and metrics related to many datasets, also performs coreference resolution. The algorithm introduced by the authors incrementally uses the same identifier for pairs of URIs (linked by the `owl:sameAs` property) defining the same resource. The authors show that by leveraging content or graph metrics, erroneous equivalence links can be detected. Their approach relies on the data provided by (Schmachtenberg et al., 2014) that are crawled from the LOD Cloud with LDSpider,[4] on various datasets: Yago, datahub.io, DBpedia v3.9, Wikidata, Freebase and LinkLion. Unfortunately, the number of coreferences that this service can offer is sometimes limited.

The sameAs.cc dataset accessible through the SPARQL endpoint[5](Raad et al., 2020) exploits the LOD-a-Lot dataset (Fernández et al., 2017) and the Louvain algorithm (Blondel et al., 2008) for detecting communities, an approach that leads them to identify errors between `owl:sameAs` relations. This approach succeeds in extracting more equivalence links than in the previous work, LODsyndesis (Mountantonakis and Tzitzikas, 2018). Different human annotators evaluated a subset of 200 `owl:sameAs` relations based on their descriptions to assess the relevance of these relations based on different thresholds of degrees of error. They apply their findings on these thresholds to cluster equivalence links that relate to the former U.S. president *Barack Obama*.[6] providing a valuable benchmark. One of the disadvantages of using the Louvain algorithm is that it must be run several times on the complete set of alleged equivalences to get the "best clusters" with no ensurance of reaching the global maximum of modularity, as it is a greedy and non-deterministic method.

Our approach differs from previous work in that it is not performed on a locally stored dataset but dynamically and online, on the SPARQL endpoints of many datasets. The list of considered endpoints is open and can be extended at will with new public and private endpoints as well as their description relevant to the application. Moreover, it uses other relations than `owl:sameAs` to define equivalence relations between different resources since we also rely on the properties `owl:InverseFunctionalProperty` and `owl:FunctionalProperty`. We also propose several techniques to curate the equivalences found on the fly and the properties declared as `owl:InverseFunctionalProperty` and `owl:FunctionalProperty`. We did this after noticing their definitions on datasets may differ from what was intended by the creators of the ontologies (see Section 5). Our algorithm also relies almost exclusively on SPARQL queries, ensuring a high compatibility with different engines and a declarativity that brings flexibility and extensibility in the sources and rules considered. The idea is to provide and evaluate a mechanism that could be implemented on top of any SPARQL engine and customized to any application.

3 SAMELIVE APPROACH: ALGORITHM AND TASK DEFINITION

Let $S = s_1, ..., s_k$ be the set of k seed URIs s_i for which we want to obtain equivalent URIs. Some of these seed URIs may be equivalent, i.e., they share a `owl:sameAs` relationship, which will result in faster convergence of the algorithm. We show this in the

[1] PREFIX ov:<http://open.vocab.org/terms/>

[2] https://web.archive.org/web/20170510073633/http://kakapo.dcs.qmul.ac.uk/ontology/musim/0.2/musim.html

[3] https://demos.isl.ics.forth.gr/lodsyndesis/

[4] https://github.com/ldspider/ldspider

[5] http://sage.univ-nantes.fr/see/sameAs

[6] https://github.com/raadjoe/obama-lod-identity-analysis

case study described in sub-section 6.3. First we need to identify a set of endpoints $N = E_1, ..., E_m$ that we will query to discover equivalent URIs. The process to calculate the closure of the equivalent relations is a greedy incremental algorithm that iterates on a growing collection of URIs linked by equivalence statements until no more new statement can be obtained from the set of endpoints N. At each iteration we also test an extensible set of declarative heuristics to detect if an equivalence is an error and avoid its transitive propagation.

Moreover, to ensure portability, distributability, and federability, the algorithm we have written is primarily in SPARQL 1.1, with calls in SPARQL 1.0[7] when we contact online SPARQL endpoints. Many endpoints on the Linked Open Data still do not support SPARQL 1.1, but the functionalities of this language allow us on a local endpoint to process properties of type (inverse) functional as well as to perform curation of `owl:sameAs` relationships and to provide statistics on the extracted data. We also trace all the steps of the algorithm by the generation of several named graphs and with the help of a small dedicated vocabulary introduced in this section and summarized in Table 1. We use a set of classes and properties to represent the input and output of each iteration. Together with the named graphs they capture the state of each iteration and the provenance of our results.

In the rest of this section, anchors in footnotes refer to labelled functions in the source code. The initialization of the algorithm consists of:

1. identifying datasets and their corresponding SPARQL endpoints and populating the named graph `same:N` with their description and checking their availability (see Section 4, done with 8 SPARQL queries, 3 of them being used for data cleaning);[8]
2. populating the named graph Q_0 with the set S of seed URIs for which we want to obtain equivalent URIs. These seeds URI are typed as `same:Target` (done with 1 SPARQL query);[9]
3. identifying and storing the definitions of `owl:InverseFunctionalProperty` and `owl:FunctionalProperty` in the named graph `kg:default`: in a preselection step the (inverse) functional properties identified from N are placed in the named graph `same:Properties` alongside with their namespaces to deference them. If they cannot be dereferenced they are stored in the named graph `same:NotDeferencedProperties` to be put to a vote (see the sub-section 5.1, done with 12 SPARQL queries, 2 of them being run in a loop).[10] This step refers to the block corresponding to the first "if" in algorithm 1.

The core of the algorithm (mainly inside the "while" loop of algorithm 1) consists of iterating on a growing collection of named graphs Q_i for which at each iteration i:

1. we query Q_{i-1} from the previous iteration for instances of `same:Target` that are not instances of `same:Rotten` (see the Figure 1), and if the result is empty, we stop (done with 1 SPARQL query);[11]
2. for each target URI we query the available endpoints in `same:N` for `owl:sameAs` relationships (done with 1 SPARQL query);[12]
3. for each target URI we query the available endpoints in `same:N` for `owl:InverseFunctionalProperty` and `owl:FunctionalProperty` concerning them and infer `owl:sameAs` relations (done with 2 SPARQL queries);[13]
4. the obtained `owl:sameAs` are added in a specific named graph with a name based on the endpoint and the iteration in which the `owl:sameAs` relationship was identified;
5. we check that the owl:sameAs relationships conform to some rules (described in the sub-section 5.2.1) and if it is not the case, identified instances of `same:Target` become of type `same:Rotten` and are placed in the named graph Q_{-1} before their relationships are removed: incoming and outgoing `owl:sameAs` relationships from a `same:Rotten` are deleted. This also applies to its `owl:InverseFunctionalProperty` and `owl:FunctionalProperty`, and the outgoing `same:Target` from a `same:Rotten` are deleted if it has no other `same:Target` incoming relationship (done with 5 SPARQL queries, 3 of them being used to remove relationships);[14]
6. the new resources are added to named graph Q_i.

[7] https://www.w3.org/TR/rdf-sparql-query/

[8] Functions $N1$ to $N3$ load information about datasets, $CN1$ is responsible for data cleaning, $A1$ is responsible for checking endpoint availability.

[9] Function $P1$ populates Q_0 with seed URIs.

[10] Function $G-(I)FP1$ retrieves the properties known as (inverse) functional, $LDD-(I)FP1$ attempts to deference and load the RDF document of properties from their namespaces, $LDS-(I)FP1$ stores as information that an endpoint has the schema of a property, $V-(I)FP1$ performs the voting on the type of the property.

[11] Function $T1$ gets the same:Target resources in Q_{i-1}.

[12] Function $S1$ retrieves `owl:sameAs` relationships.

[13] Function $(I)FP1$ retrieves the actual instances of (inverse) functional properties and $(I)FP2$ infers `owl:sameAs` relationships from them.

[14] Function $R1$ refers to the rule #1 used to detect `same:Rotten`, respectively $R2$ refers to the rule #2, $RC1$ is used to remove relationships ; details in Section 5.2.

The stop condition is that the resources to be explored are exhausted, i.e., there are no more resources to explore for the current iteration i and Q_i is empty. A last check with (c.f., the rule #2, sub-section 5.2.1) is necessary to ensure that there is no relationship between two resources sharing the same authority (resources obtained at i). The pseudo-code of SameLive is represented by the algorithm 1.

Algorithm 1: Online resolution of identity with the SameLive algorithm.

```
Initialize Vocabulary // Also initialize Q_-1
Initialize N // Availability is checked
Initialize Q_0
// Variable to consider or not
   (inverse) functional properties
properties_condition = {True, False}
if properties_condition then
    Search properties typed as (inverse) functional
      over N
    Deferenciation of these properties
    Search schema of properties not deferenced
      over N
    Voting for the properties not deferenced
i = 1
L = length(Q_0)
while L != 0 do
    Initialize Q_i
    Q_i += Retrieve owl:sameAs relations of Q_i-1
      over N
    if properties_condition then
        Retrieve triples comporting Q_i-1 who
          have a property typed typed as (inverse)
          functional over N
        Q_i += Infer owl:sameAs relations with the
          triples previously extracted over N
    // Curation rules have an impact on
       all Q_i
    Q_-1 += Apply curation rules
    i += 1
    L = length(Q_i-1)
Q_-1 += Apply curation rule #2
```

4 EXTRACTION AND INTEGRATION OF ENDPOINTS' INFORMATION

The first step is to identify the datasets that can contribute to solving this problem. Various strategies exist for this purpose such as relying on search engines, using previously crawled RDF data (i.e., using the RFC 8615,[15] etc.) or even catalogs referencing these datasets. To achieve our means, we rely on famous catalogs listing datasets, using their metadata as well as the URLs for their SPARQL endpoints. These catalogs are regularly updated so we can keep up with the latest updates of the Web of Data. Moreover this approach supports the addition of new catalogs at any time, including private endpoints.

However, just because a dataset is referenced in a catalog does not mean it is available. To define if a SPARQL endpoint is available or not, we rely on the EndS ontology (Endpoint Status Ontology)[16] which is an extension of VoiD.[17] We use in particular the property `ends:statusIsAvailable`. This ontology is involved in the description of the different datasets retrieved from the catalogs detailed in the following sub-sections. For each source we create a distinct named graph and then combine all unduplicated results (i.e., different SPARQL endpoint URL) into one initial named graph `same:N` containing all our sources. From the different catalogs we present, we only rely on the asserted availability of endpoints from YummyData. The other catalogs listed here do not have regular updates on this property and we perform the availability ourselves.

4.1 voiD store

We rely on the voiD store[18] and query it for instances of `void:Dataset` with their `void:sparqlEndpoint` to access them. To avoid duplicating entries for a same dataset -as datasets can be referred with several types in the voiD store- or incomplete results we limit ourselves to the extraction of resources of type `void:Dataset` and we ensure to not retrieve blank nodes. We also check that these resources have a title (property `dcterms:title`)[19] and, of course, a SPARQL endpoint (property `void:sparqlEndpoint`). Any doubloon is eliminated (i.e., datasets using the same SPARQL endpoint), datasets can be represented several times in the voiD store. The retrieval of the information about endpoints and the removal of the doubloons is carried out in two steps to overcome the limitations of the voiD store SPARQL endpoint.[20]. The endpoint of the voiD store has recently been closed due to the decision of the service maintainers, however our approach still works with the other catalogs and we pro-

[15] https://tools.ietf.org/html/rfc8615

[16] PREFIX ends: <https://labs.mondeca.com/vocab/endpointStatus/>, archive link: https://web.archive.org/web/20210302021149/https://labs.mondeca.com/vocab/endpointStatus/

[17] https://www.w3.org/TR/void/

[18] http://void.rkbexplorer.com

[19] PREFIX dcterms: <http://purl.org/dc/terms/>

[20] http://void.rkbexplorer.com/sparql

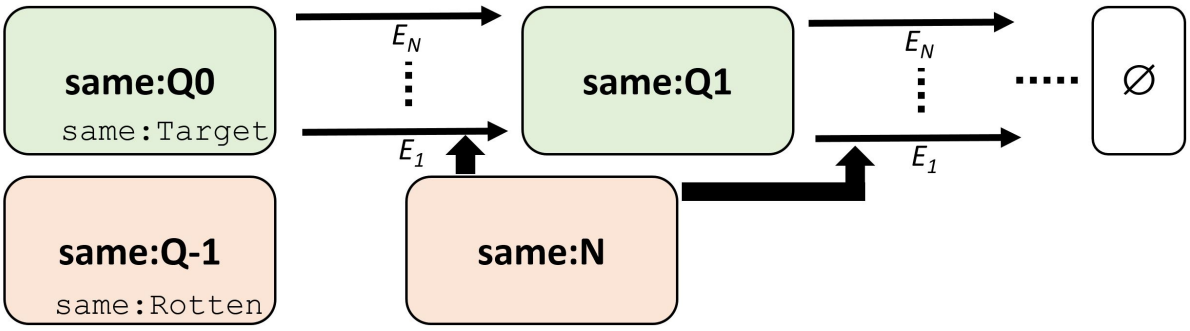


Figure 1: Workflow diagram of the steps followed to retrieve equivalent links from the starting resources in Q_0 with the available endpoints in `same:N`.

Table 1: Main vocabulary introduced for the equivalence discovering algorithm.

Element	Main type	Role
same:Rotten	owl:Class	Resource that is source of potentially erroneous relationships.
same:Target	owl:Class	Targeted resource for discovering equivalence relationships.
same:hasAuthority	owl:DatatypeProperty	Authority of the resource.
same:hasIteration	owl:DatatypeProperty	Iteration of the equivalence relationships algorithm during which the named graph was generated.
same:hasNamespace	owl:DatatypeProperty	Namespace of the resource.
same:statementInDataset	rdf:ObjectProperty	Points to the void:Dataset that a statement is a part of.
same:votingType	owl:ObjectProperty	rdf:type of the resource determined after voting.
same:N	rdfg:Graph	Named graph that contains information about the void:Dataset resources.
same:Properties	rdfg:Graph	Named graph that contains asserted owl:InverseFunctionalProperty and owl:FunctionalProperty.
same:PropertiesNotDeferenced	rdfg:Graph	Named graph that contains not deferenced properties after the LOAD clause.
same:Q-1	rdfg:Graph	Named graph that contains the same:Rotten resources.
same:$Q0$	rdfg:Graph	Named graph that contains the starting same:Target.
same:Qi	rdfg:Graph	Named graph that contains the same:Target retrieved at the i iteration of the equivalence relationships algorithm.

vide code to integrate its data if someone decides to continue maintaining this service. Our results were obtained while this service was still in operation. A large majority of the core datasets of the Linked Open Data are also referenced by the LODCloud.

4.2 LODCloud

The main purpose of the LODCloud[21] website is to provide a diagram of the LOD cloud. The JSON data used to build it are available[22] and we translate selected parts of them in RDF as different data are represented within this JSON document. We first check if a dataset has an entry related to a SPARQL endpoint ("sparql" field) in which case we include it. If we find information about a voiD page, we store this data as well. However, some of the information contained in the lod-cloud is not always up to date: an endpoint marked as unavailable may be in fact available at the time of our query and vice-versa. Therefore, before executing our approach we check the availability of the endpoints to avoid waiting unnecessarily for a response from an unavailable endpoint.

4.3 YummyData

YummyData[23] (Yamamoto et al., 2018) is a site referencing and monitoring various SPARQL endpoints in the biomedical domain. We first get the URL leading to the SPARQL endpoint of the datasets from the JSON data obtained from their API. Then we check if the datasets have a VoiD annotation ('void' field equal to True) and in such case we remove the suffixes "virtuoso/sparql" or "sparql" from the URLs of the SPARQL endpoints and we add as a new suffix ".well-known/void" to refer to the VoiD page describ-

[21] https://lod-cloud.net/

[22] https://lod-cloud.net/lod-data.json

[23] https://yummydata.org/

ing the datasets.[24] For instance, to obtain the VoiD page of the Protein Ontology the URL https://sparql.proconsortium.org/virtuoso/sparql becomes https://sparql.proconsortium.org/.well-known/void).

Now that we have described how we retrieve information about SPARQL endpoints, we will get to the heart of the matter by describing the resources needed by the algorithm and some of its key steps.

5 EQUIVALENCE LINKS RETRIEVAL AND DATA CURATION

5.1 Collecting true instances of (inverse) functional properties

To establish identity, we collect relevant `owl:InverseFuntionalProperty` and `owl:FunctionalProperty` that we store in the named graph `same:Properties`.

According to OWL specifications: if a, b, c are three resources, (i)[25] if there exist inverse functional property relations $ip(a,b)$ and $ip(c,b)$ there also exists an equivalence relation $owl:sameAs(a,c)$ and (ii)[26] similarly, if there exist functional property relations $fp(a,b)$ and $fp(a,c)$ there exists an $owl:sameAs(b,c)$ equivalence relation.

However, depending on the SPARQL endpoints, the properties' definitions may vary and contain abusive usage of these types or at least usage that should be limited to a local closed world. For instance, `rdfs:label` is defined as an `owl:InverseFunctionalProperty` in the National Digital Data Archive of Hungary[27] or `agront:isPartOfSubvocabulary`[28] is defined as a `owl:FunctionalProperty` by the LusTRE endpoint.[29] Table 2 provides some statistics we computed about this problem. The following steps (sections 5.1.1 to 5.1.3) refer to the block corresponding to the first "if" in algorithm 1 and the item 3 in the first list in Section 3.

5.1.1 Initialization

We first query all available endpoints (property `ends:statusIsAvailable` set to true) in the named graph `same:N` to return their properties of type `owl:InverseFunctionalProperty` and `owl:FunctionalProperty`. A filter is performed on the properties defined as blank nodes. Then, these data are stored as RDF* triples (Hartig and Thompson, 2014) in the named graph `same:Properties` and we keep as information the dataset from which they come with the `same:statementInDataset` property. These triples are stored in RDF* to record the statements typing properties together with the provenance dataset in which they were found, i.e.:

```
<<foaf:firstName a owl:FunctionalProperty>>
same:statementInDataset <http://uriburner.com/>
```

We then perform two types of curation: curation by schema and curation by voting. Properties for which the definition could not be deferenced or found in the SPARQL endpoints are excluded.

5.1.2 Curation by schema

We ensure that the use of these properties corresponds to what was intended by the creators of the ontologies is to extract the schemas of these properties by deferencing them. To do so, we use the namespaces of these properties which we have inserted into our local triplestore in the named graph `same:Properties` using the property `same:hasNamespace` and perform the SPARQL clause:

```
LOAD SILENT <namespace> INTO GRAPH kg:default;
```

Using namespaces to retrieve RDF documents is more efficient since it avoids downloading multiple times the same document where multiple properties of type `owl:InverseFunctionalProperty` and `owl:FunctionalProperty` are defined. These documents are then loaded into the named graph `kg:default`. This first step already corrects some assertions about properties (see Table 2).

5.1.3 Curation by voting

In a second step, we handle properties for which no schema could be deferenced. We have placed these properties in the graph named `same:NotDeferencedProperties` to distinguish them from the others. We perform queries on the available endpoints in the set N.

After identifying whether the endpoints have the schemas of a property, we count the number of endpoints including the schemas against the property

[24] see VoiD documentation on "discovering": https://www.w3.org/TR/void/#well-known.

[25] https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/#Inverse-Functional_Object_Properties

[26] https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/#Functional_Object_Properties

[27] http://lod.sztaki.hu/sparql

[28] http://aims.fao.org/aos/agrontology#isPartOfSubvocabulary

[29] the endpoint http://linkeddata.ge.imati.cnr.it:8890/sparql includes the EARTh -Enviromental Applications Reference THesaurus- and ThIST -Italian Thesaurus of Earth Sciences- datasets)

types so that we can perform a vote on the most accepted definition of a property. The condition for assigning the type (property same:votingType) owl:InverseFunctionalProperty and/or owl:FunctionalProperty is that a property must be defined as such, by at least half of the SPARQL endpoints that store a schema for the given property. The statistics to compute the type of these properties are stored in the named graph same:PropertiesNotDeferencedStatistics. For this purpose we declared several properties such as same:nbOfTimesDefinedAsFunctionalProperty and same:inNbOfDatasetWithSchema. By this means, we are able to identify for example the property sparql:endpoint[30] as an owl:InverseFunctionalProperty and the property semsci:CHEMINF_000009[31] as a owl:FunctionalProperty.

From the set of curated properties, we can infer owl:sameAs relationships of the same:Target of the current *Qi* with the available endpoints of same:N.

5.2 Extraction of owl:sameAs properties

Now that we have inspected and corrected the type of the owl:InverseFunctionalProperty and owl:FunctionalProperty properties, it is possible to infer owl:sameAs relationships. To do this, we extract the owl:sameAs relationships for the same:Target resources and curate them by following an iterative mechanism. The goal being not to question the endpoints twice about the same resource same:Target for its owl:sameAs relationships. For this purpose, these resources are stored in different named graphs same:Qi at each iteration *i*. We query the available endpoints in *N* with the same:Target of the current iteration (in same:Qi) and we store the resulting equivalent URIs in the next named graph same:Qi+1. A filter clause ensures that we do not store an already existing same:Target in the named graph same:Qi+1. The stop condition of the algorithm is to exhaust all instances of same:Target. This step refers to the block corresponding to the "while" loop of algorithm 1.

Working with live endpoints of the World *Wild* Web, we pay attention to technical details such as the fact that we have to query some endpoints with only URIs that include ASCII characters since we have identified some SPARQL endpoints that do not support non-ASCII characters.[32] We actually extract and store URIs containing non-ASCII characters but we perform a transformation using the following SPARQL clause: FILTER(!REGEX(str(?URITarget), "[^\x00-\x7F]", "i")). This point can be improved later in a future work with a more detailed description of SPARQL access points. We also extract the authority component of the URIs[33] and store them with the same:hasAuthority property. This will be used to curate owl:sameAs relationships.

5.2.1 Curation of the owl:sameAs links

We carry out the curation of owl:sameAs relationships assuming that equivalent resources with URIs defined within the same authority must explicitly be asserted as equivalent by that authority. These links were obtained through a direct owl:sameAs relationship or from a owl:InverseFunctionalProperty and a owl:FunctionalProperty. Thus, we will ensure that the resources linked by owl:sameAs relationships comply with the rules defined below.

We identified two different patterns for erroneous relationships linked through resources that we call same:Rotten. More patterns may be added to further extend the constraints that equivalent resources must meet.

For complexity reasons we split the curation between two rules: one operating on the results from the same authority obtained at distinct iterations (rule #1) and one operating on the results of the same authority at the same iteration (rule #2):

- Rule #1 : Let a: be the prefix of an authority and b: the prefix of another authority, if there is a sameAs(a:1,b:1) relationship and a sameAs(b:1,a:2) relationship and no sameAs(a:1,a:2) relationship, this rule states that b:1 is of type same:Rotten. This rule is applied regardless of the length of the path of owl:sameAs between a:1 and a:2.
- Rule #2 : Let a: be the prefix of an authority and b: the prefix of another authority, if there is a sameAs(a:1,b:1) relationship and a sameAs(a:1,b:2) relationship and no sameAs(b:1,b:2) relationship, this query states that b:1 and b:2 are both of type same:Rotten.

The Figure 2 displays two examples on which these rules are applied. Once identified as such we delete the owl:sameAs relations including a same:Rotten and the same:Target resulting from

[30] http://www.w3.org/ns/sparql-service-description#endpoint

[31] https://semanticscience.org/resource/CHEMINF_000009

[32] e.g. http://linkedlifedata.com/sparql

[33] URI = scheme:[//authority]path[?query][#fragment] in RFC 3986

these resources. Before the application of the two rules above, all resources (a:1, a:2, b:1, b:2) are of type `same:Target`. Contrarily to other methods, this approach does not require the extraction of all the alleged existing equivalence relations for processing their quality and it also trims as soon as possible the exploration of bad quality equivalences and their transitive closure.

6 EXPERIMENTAL PROTOCOL AND EVALUATION

6.1 Quantitative Evaluation on Linked Open Data

Table 2 shows statistics about the properties of type `owl:InverseFunctionalProperty` and `owl:FunctionalProperty` found online and on which we applied the process described in section 5.1 to collect true instances of these types. Only 22% of the RDF documents queried that a priori contained properties of type `owl:InverseFunctionalProperty` or `owl:InverseFunctionalProperty` could be loaded. Approximately 17% of the properties claimed of type `owl:FunctionalProperty` were verified as such by voting or dereferencing, while for the properties of type `owl:InverseFunctionalProperty`, they have been verified as such in 60% of the cases. The large majority of properties of type `owl:InverseFunctionalProperty` have been deferenced, while about half of the properties of type `owl:FunctionalProperty` were identified by voting.

6.1.1 Protocol, Dataset and Baselines

The different experiments were conducted on an HP EliteBook 840 G2, 2.6 hHz, 16 GB RAM with a virtual environment under Python 3.8.5 and the software Corese Semantic Web Factory[34] (Corby and Zucker, 2010) version 4.1.6d deployed locally. Corese is used as a local triplestore on which we mainly use SPARQL 1.1 Query and Update features. We evaluated our approach on the `Barack Obama` identity links knowledge graph developed by (Raad et al., 2020).[35] Regarding the initialization, we declared for our algorithm the target URI dbr:Barack_Obama[36] (a `same:Target`) in the named graph `same:Q0`. The different closures on the *Barack Obama* entity with which we compare ourselves (with *sameAs.cc* (Raad et al., 2020), *LODsyndesis* (Mountantonakis and Tzitzikas, 2018) and *sameas.org* (Jaffri et al., 2008)) and our approach, *SameLive*, are the following:

- *Ground truth*: The manual annotation of the closure of the owl:sameAs extracted from the LOD-a-lot dataset distributed into 8 identity sets (Barack Obama, Obama's Presidency, Obama's Presidential Transition, Obama's Senate Career, Obama's Presidential Centre, Obama's Biography, Obama's Photos, Black President). The `undetermined URIs` included in the identity cluster about *Barack Obama* are essentially URIs for which we do not have enough semantics to confidently annotate them.
- *sameAs 0.99*: Results after removing the relations with an error degree greater than 0.99 with the method used by sameAs.cc where the error degree is based on the communities resulting from the Louvain algorithm. This approach leads to two identities sets B_1 and B_2, and enabled the separation of URIs referring to the *Obama's presidency* and his *presidential transition* from the other identity sets. However, these two sets are still inconsistent since they do not allow to perform a closure on a single real world entity.
- *sameAs 0.4*: Results after removing the relations with an error degree greater than 0.4 in the method of sameAs.cc. This approach leads to 219 identity sets (C_1 to C_{219}) with only one identity set C_1 with non-singleton URIs.
- *sameAs.org*: Results were obtained through the API of sameAs.org,[37] and we corrected URI encoding issues to avoid counting them as new ones.
- *LODsyndesis*: To detect errors, both clustering with similarity function (content based detection) and shortest path between a pair of instances (graph based detection) are used. Results were obtained through the API of LODsyndesis,[38] and we corrected their encoding issues too.
- *SameLive*: Our closure performed on the data extracted from the SPARQL endpoints indexed by the voiD Store, the LODCloud and the Yummy-Data websites. O_1 only includes resources of type `same:Target`, O_2 contains both resources of types `same:Target` and `same:Rotten`

[34] https://project.inria.fr/corese/

[35] https://github.com/raadjoe/obama-lod-identity-analysis

[36] PREFIX dbr: <http://dbpedia.org/resource/>

[37] http://sameas.org/rdf?uri=http://dbpedia.org/resource/Barack_Obama

[38] https://demos.isl.ics.forth.gr/lodsyndesis/rest-api/objectCoreference?uri=http://dbpedia.org/resource/Barack_Obama

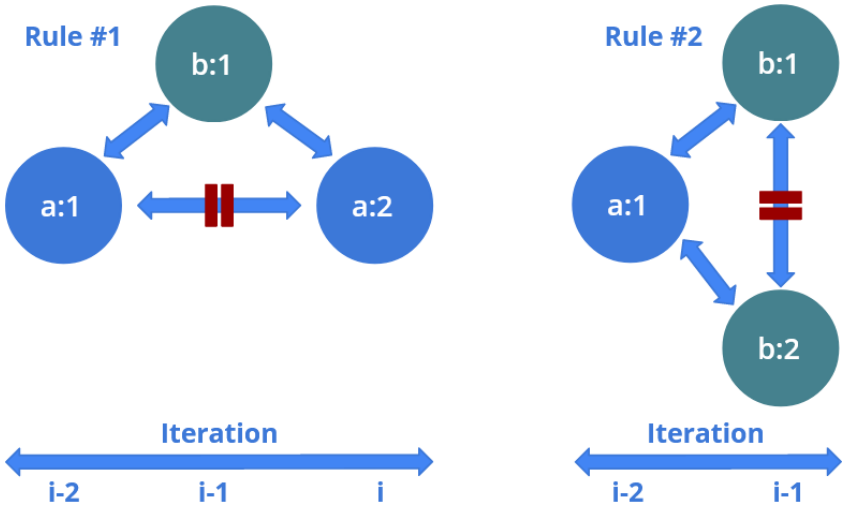


Figure 2: Examples of resources identified as `same:Rotten` (green dots) in a path of `owl:sameAs` relations with the rules mentioned in the sub-section 5.2.1. The red pause icon indicates that there is no `owl:sameAs` relationship between two resources.

Table 2: Statistics linked to the properties of type `owl:InverseFunctionalProperty` and `owl:FunctionalProperty`. The incorrect properties are stated as such after loading the RDF document defining them.

Category	Number of elements
Properties asserted as owl:InverseFunctionalProperty	3784
Properties asserted as owl:FunctionalProperty	13298
Not deferenced properties among those extracted	14192
Incorrect owl:InverseFunctionalProperty properties	10
Incorrect owl:FunctionalProperty properties	35
Properties participating in the voting process	1220
Properties voted as owl:InverseFunctionalProperty	123
Properties voted as owl:FunctionalProperty	1111
Properties voted as not owl:InverseFunctionalProperty	1097
Properties voted as not owl:FunctionalProperty	109
Final number of owl:InverseFunctionalProperty	2288
Final number of owl:FunctionalProperty	2261
Number of loaded RDF documents	168
Number of missing RDF documents	781

6.1.2 Closure evaluation

Table 3, based on the work of (Raad et al., 2020), reports the results of our approach compared to sameAs.cc, LODsyndesis and sameAs.org on the `Barack Obama` identity links knowledge graph.

6.1.3 Discussion

After three iterations of the algorithm, we reached the closure of our solution. The closure on dbr:Barack_Obama by considering only `owl:sameAs` relationships takes 40 minutes, and considering the `owl:InverseFunctionalProperty` and `owl:FunctionalProperty` properties takes 21 hours and a half with a regular laptop setup described in section 6.1.1. The execution time comes mainly from the endpoint querying. We intend to look at approaches to improve this with timeouts and further parallel querying. We are currently working on a cluster version of SameLive for this purpose. In terms of statistics that represent 1 starting URI `same:Target` (in $Q0$), 130 in $Q1$, 9 in $Q2$, 1 in $Q3$ and 53 `same:Rotten` in Q_{-1}.

Assessing our results on this graph presents a clear disadvantage for our approach because it involves a snapshot of the 2015 Linked Open Data (LOD). As a result, links that have disappeared are not included in our approach. However these are the best baselines as far as we know and our crawling-free online approach still obtains slightly lower results but comparable to the best approach, the *sameAs 0.4* proposed by (Raad et al., 2020). Moreover, if we focus on the advantages of SameLive, the online na-

Table 3: Comparison of the owl:sameAs closures on dbr:Barack_Obama.

	Ground truth	sameAs 0.99		sameAs 0.4	sameAs.org	LODsyndesis	SameLive	
Real World Entity	A_1	B_1	B_2	C_1	S_1	L_1	O_1	O_2
Barack Obama	260	260	0	120	240	19	105	116
Other Real World Entity	78	10	68	0	22	0	0	0
New URIs outside A_1	0	0	0	0	413	14	27	67
Undetermined URIs	102	92	10	1	32	4	9	11
Identity Sets	1	2		219	1	1	1	1
Total URIs in Identity Set	440	362	78	121	707	37	141	194

ture of it, we are able to identify a total of 141 equivalent resources of type same:Target and 53 same:Rotten (subtraction of sets O_2 and O_1 on the total of URIs), and 67 new URIs about *Barack Obama* compared to the LOD-a-Lot dataset. As an example, we identified as same:Rotten URIs coming from URIBurner that include resources of dubious quality such as "The Irishman" on Netflix.[39] 11 resources of type same:Rotten are considered as belonging to the identity set of *Barack Obama* (subtraction of sets O_2 and O_1 on the entity set about *Barack Obama*).

The SPARQL query we use to detect potential errors eliminates redirect links in a dataset if they are not declared as owl:sameAs in it (or by extension if there is no owl:InverseFunctionProperty or owl:FunctionalProperty relationship in the same dataset). To increase this coverage with our approach, we would have to include specific properties to redirects such as the property dbo:wikiPageRedirects[40] with DBpedia).

6.2 Qualitative Evaluation on Specific Examples

From the Barack Obama identity links knowledge graph, one can notice that: some of the resources evaluated as being of the same nature as dbr:Barack_Obama are no longer valid,[41] URLs redirecting to valid resources are considered valid,[42] while other more questionable resources are also considered valid.[43]

Depending on the application, it may be unnecessary to obtain such a degree of confidence for obtaining equivalence links as proposed by *sameAs 0.99*, but in the case where one really wants to obtain owl:sameAs relations, only *sameAs 0.4*, *LODsyndesis* and our approach provide an answer. With the graph of identity links about *Barack Obama* only entities more or less related to Obama exist. However this is not the case for all the graphs obtained by extracting owl:sameAs relationships (i.e., previous extractions present on the English DBpedia endpoint linked dbr:Berlin to dbr:Tirana, dbr:Gaspé_Peninsula, dbr:Jersey_City,_New_Jersey, dbr:Point_Reyes_National_Seashore, dbr:Flint,_Michigan, and this is still the case on sameAs.org). Thus, even if we lower our similarity expectations, the *sameAs 0.99* approach does not necessarily guarantee that on such graphs its results do not keep wrong relationships.

Our system, unlike sameAs.cc, discriminates redirects to a resource if it does not have a direct owl:sameAs relationship with that resource. This point may have its advantage in error detection. However the public benchmark on which we are evaluating discriminates against this position.

Another important point is that two resources belonging to the same identity cluster may result in the generation of different equivalence links depending on the starting resources. The reason for this is that we do not process the whole graph of equivalences, the greedy algorithm starts from the set S of seeds and stops as soon as possible to avoid propagating errors or querying the endpoints more than necessary.

6.3 Application Evaluation on Specific Use Case

The source code of the project and its instructions are available at https://github.com/Wimmics/SameLive and the algorithm vocabulary at https://ns.inria.fr/same/same.owl. Two modes of the algorithm are available to consider or not the owl:InverseFunctionalProperty and owl:FunctionalProperty. We also performed an evaluation on a real application: the integration of

[39] e.g., http://linkeddata.uriburner.com/about/id/entity/https/www.nytimes.com/2019/12/06/business/media/irishman-scorsese-netflix-ratings.html?smid=tw-nytimes&smtyp=cur#entity_534366

[40] PREFIX dbo: <http://dbpedia.org/ontology/>

[41] e.g., this is not visible on web.archives.org and is now an advertising website: http://www.ontosearch.com/2008/01/identification/EID-3b6e3fb1eb4bef8e669277e73d2e7d56

[42] e.g., http://dbpedia.org/resource/44th_president_of_the_united_states_of_america

[43] e.g., http://dbpedia.org/resource/B-Rock_%22The_Islamic_Shock%22_Hussein_Superallah_Obama

the knowledge graphs obtained from different named entity recognition and linking methods applied to the same corpus. In this type of usecase, to consolidate the knowledge graph and support further processing and visualization, it is important to detect that two URIs extracted by two different annotators are in fact identifying the same resource. We tested our approach on the public dataset CovidOnTheWeb (Michel et al., 2020), where the authors extract named entities but do not propose equivalence links between the URIs produced by different annotators. As an example we report here on the result obtained on the article "COVID-19: what has been learned and to be learned about the novel coronavirus disease"[44] (Yi et al., 2020) and more precisely on the 312 distinct entities from DBpedia and Wikidata extracted from it with semantic annotators.

By exploring only the owl:sameAs relationships on these 312 starting URIs (in $Q0$), the algorithm ends after 6 iterations with: 1526 `same:Target` in $Q1$, 3762 in $Q2$, 1797 in $Q3$, 78 in $Q4$, 71 in $Q5$ and 2 in $Q6$. We computed a total of 6001 `same:Rotten` in Q_{-1}, correcting errors such as, for instance, the confusion between URIs coming from the French chapter of DBpedia identifying the kidney vs. a specific portion of it called the "distal convoluted tubule".[45] Among the 312 seeds URIs, 32 are determined as equivalent. Some of these 32 resources considered as equivalent are not available in sameAs.cc, sameAs.org and LODsyndesis. This is the case for example for: the respiratory syndrom caused by SARS coronavirus 2 in DBpedia and Wikidata,[46] and the strain of the COVID-19 itself in these two sources.[47] An interesting fact is that so far these resources speaking about the COVID-19 do not have any `owl:sameAs` relationship in the English chapter of DBpedia.

7 CONCLUSION

We proposed a method to identify `owl:sameAs` relationships relying on the online SPARQL querying of distributed datasets and using heuristic rules to correct results. We also exploit and inspect the quality of `owl:InverseFunctionalProperty` and `owl:FunctionalProperty` relationships, using the definitions given by endpoints and a voting approach. We show that a heuristic approach can retrieve high quality equivalence links without requiring the extraction of all the alleged existing equivalence relations. In addition, it is possible to use other algorithms (community detection, similarity function, graph-based metric, etc.) in addition to/instead of the curation rules we have implemented.

Because our algorithm works online this also exposes it to return different results for the same input, for example, if a SPARQL endpoint does not answer. Inversely, our method has the advantage of giving the user the possibility to include or exclude endpoints (public or private) on the fly, and thus to include locally stored datasets as long as a SPARQL endpoint is deployed.

It may be interesting to include other dataset catalogs, such as the ones using the CKAN API.[48] Related to this, integrating SPARQL Micro-Services (Michel et al., 2018) or other mapping approaches on top of these catalogs would allow us to query them directly with SPARQL instead of manipulating beforehand their data with another language. The use of such techniques could also help us extend our method to include non-RDF datasets.

Extracting `owl:InverseFunctionalProperty`, `owl:FunctionalProperty` and `owl:sameAs` relationships is not enough for some datasets. For instance Wikidata formalizes equivalence relationships with identifiers (typed as external identifier `wikibase:ExternalId` using `wikibase:propertyType`[49]) leveraging URI patterns. Also, although they seem to be rarely used, we intend to study the inclusion of `owl:hasKey` to have a modular and extensive set in the analysis of equivalence links. In addition we plan to consider redirects -including different scheme component of the URIs such as HTTP/S. All these extensions could be addressed by adding additional rules, some of which could be the result of mining and learning approaches. Deferencing resources deemed as `same:Rotten` and comparing them to `same:Target` with a similarity metric is also a direction to obtain more results. Exploiting the underlying semantics (e.g., `owl:differentFrom`, `owl:AllDifferent`...) of the resources is also worth exploring.

Finally, we intend to follow different leads to improve the performance of our approach in terms of speed, from query optimization (in particular for the `owl:InverseFunctionalProperty` and `owl:FunctionalProperty` properties) and further parallel querying. We will also study the software

[44] http://ns.inria.fr/covid19/0eadf5a901c0d89fad2c202990056556be103e12

[45] e.g., http://fr.dbpedia.org/resource/Tubule_distal and http://fr.dbpedia.org/resource/Rein

[46] e.g., http://dbpedia.org/resource/Coronavirus_disease_2019 and http://www.wikidata.org/entity/Q84263196

[47] e.g., http://dbpedia.org/resource/Severe_acute_respiratory_syndrome_coronavirus_2 and http://www.wikidata.org/entity/Q82069695

[48] https://docs.ckan.org/

[49] PREFIX wikibase: <http://wikiba.se/ontology#>

and hardware architecture needed to provide a web service with a caching system. We plan also to further exploit the monitoring capabilities of SPARQL, by using for example the PROV-O ontology[50] to better track the provenance of results, this could be used in particular for `owl:sameAs` relationships stored in specific named graphs (see Section 3, item 4 in the second list). We also want to exploit timestamps to, among other things, timely re-run queries executed a long time ago or to query an endpoint that was previously unavailable.

REFERENCES

Allemang, D., Hendler, J., and Gandon, F. (2020). *Semantic Web for the Working Ontologist*. Number 3. ACM.

Blondel, V. D., Guillaume, J.-L., Lambiotte, R., and Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008.

Corby, O. and Zucker, C. F. (2010). The kgram abstract machine for knowledge graph querying. In *Web Intelligence and Intelligent Agent Technology (WI-IAT)*, volume 1, pages 338–341. IEEE.

Fernández, J. D., Beek, W., Martínez-Prieto, M. A., and Arias, M. (2017). Lod-a-lot. In *International semantic web conference*, pages 75–83. Springer.

Gandon, F. (2018). A Survey of the First 20 Years of Research on Semantic Web and Linked Data. *Revue des Sciences et Technologies de l'Information - Série ISI : Ingénierie des Systèmes d'Information*.

Halpin, H., Hayes, P. J., McCusker, J. P., McGuinness, D. L., and Thompson, H. S. (2010). When owl: sameas isn't the same: An analysis of identity in linked data. In *International semantic web conference*, pages 305–320. Springer.

Hartig, O. and Thompson, B. (2014). Foundations of an alternative approach to reification in rdf. *arXiv preprint arXiv:1406.3399*.

Jaffri, A., Glaser, H., and Millard, I. (2008). Managing URI synonymity to enable consistent reference on the semantic web. In *Proceedings of the 1st IRSW2008 International Workshop on Identity and Reference on the Semantic Web*.

Michel, F., Faron-Zucker, C., and Gandon, F. (2018). SPARQL micro-services: Lightweight integration of web apis and linked data. In *Workshop on Linked Data on the Web co-located with The Web Conference 2018, LDOW@WWW 2018*.

Michel, F., Gandon, F., Ah-Kane, V., Bobasheva, A., Cabrio, E., Corby, O., Gazzotti, R., Giboin, A., Marro, S., Mayer, T., et al. (2020). Covid-on-the-web: Knowledge graph and services to advance covid-19 research. In *International Semantic Web Conference*, pages 294–310. Springer.

Mountantonakis, M. and Tzitzikas, Y. (2018). Scalable methods for measuring the connectivity and quality of large numbers of linked datasets. *Journal of Data and Information Quality (JDIQ)*, 9(3):1–49.

Raad, J., Beek, W., van Harmelen, F., Wielemaker, J., Pernelle, N., and Saïs, F. (2020). Constructing and cleaning identity graphs in the lod cloud. *Data Intelligence*, 2(3):323–352.

Schmachtenberg, M., Bizer, C., and Paulheim, H. (2014). Adoption of the linked data best practices in different topical domains. In *International Semantic Web Conference*, pages 245–260. Springer.

Yamamoto, Y., Yamaguchi, A., and Splendiani, A. (2018). YummyData: providing high-quality open life science data. *Database*, 2018.

Yi, Y., Lagniton, P. N., Ye, S., Li, E., and Xu, R.-H. (2020). Covid-19: what has been learned and to be learned about the novel coronavirus disease. *International journal of biological sciences*, 16(10):1753.

[50] https://www.w3.org/TR/prov-o/