SUBMASSIVE: Resolving Subclass Cycles in Very Large Knowledge Graphs

Shuai Wang, Peter Bloem, Joe Raad, and Frank van Harmelen

Knowledge Representation and Reasoning Group, Department of Computer Science,
Vrije University Amsterdam, Amsterdam, the Netherlands
{shuai.wang| p.bloem| j.raad| frank.van.harmelen}@vu.nl

Abstract. Large knowledge graphs capture information of a large number of entities and their relations. Among the many relations they capture, class subsumption assertions are usually present and expressed using the `rdfs:subClassOf` construct. From our examination, publicly available knowledge graphs contain many potentially erroneous cyclic subclass relations, a problem that can be exacerbated when different knowledge graphs are integrated as Linked Open Data. This paper presents an automatic approach for resolving such cycles at scale using automated reasoning by encoding the problem of cycle-resolving to a MAXSAT solver. The approach is tested on the LOD-a-lot dataset, and compared against a semi-automatic version of our algorithm. We show how the number of removed triples is a trade-off against the efficiency of the algorithm. The code and the resulting cycle-free class hierarchy of the LOD-a-lot are published at `www.submassive.cc`.

Keywords: Knowledge graph refinement · LOD-a-lot · Automated reasoning

1 Introduction

Among the many relations knowledge graphs capture, subsumption relations on classes are represented as triples of axioms using `rdfs:subClassOf`. Ideally, such triples form a structure of hierarchy, a.k.a. an acyclic graph (DAG), if not considering reflexive relations. From our research, not all knowledge graphs have such an acyclic class taxonomy. In the case of very large knowledge graphs, integrating information from different sources, the cyclic relations from integrated small knowledge graphs are inherited and there is potentially erroneous cyclic information across different domains.

This paper focuses on resolving potentially erroneous cyclic relations in the aim of obtaining a reliable cycle-free hierarchy with a minimum number of removed relations. In Section 1, we first study some properties of cyclic subclass assertions (Section 1.1) and present related work (Section 1.2). Section 2 describes our algorithm for removing potentially erroneous cycles, followed by its implementation details. Finally we evaluate our system and present results in Section 3 followed by a discussion and our plan for future work in Section 4.

1.1 Preliminaries

A knowledge graph (KG) G consists of triples of relational information on entities. Such triples are in the form of (s, p, o), where s is an entity called the *subject*, p is a predicate, and o is an entity called the *object*. A relation such as class A is a subclass of class B ($A \sqsubseteq B$) is encoded as (A, `rdfs:subClassOf`, B). From a graph theory point of view, these triples form a directed multigraph with self-loops (a.k.a. reflexive relations). When studying those triples where p is restricted to rdfs:subClassOf, the subgraph G_{cat} is solely about the subsumption relations on classes. Cyclic class relations come in the form $A \sqsubseteq A$ (a reflexive relation; self-loop), or $A \sqsubseteq B$ and $B \sqsubseteq A$ (size two), or more generally $A_1 \sqsubseteq A_2 \sqsubseteq \ldots \sqsubseteq A_n \sqsubseteq A_1$ (size N).

Such cycles can be harmless. For example, reflexive cycles are simply tautologies (always true) and are therefore redundant. There are four possible inclusion relations between two classes A and B: $A \sqsubseteq B$, $B \sqsubseteq A$, $A \equiv B$ ($A \sqsubseteq B$ and $B \sqsubseteq A$), or none of them (including the case of unknown). Due to the transitivity, cycles are only correct when all classes in the cycle are equivalent and are otherwise a source of error.

In this paper, we use a hybrid approach of logical and network/graph theory methods. To avoid confusion, we clarify the terminology: we use *relation* interchangeably with edge and triple; we let *class* to be the same as *node* and *concept*; we use *graph* for *knowledge graph* or *knowledge base*. Similarly, we use *cycle* in reference to *cyclic subclass assertions*.

1.2 Related Work

According to [1], knowledge graph refinement methods can be divided into two main categories: *completing* the knowledge graph with missing knowledge, and *identifying wrongly* asserted information. This work falls into the latter category of approaches, as we attempt to resolve cyclic subclass relations by removing potentially incorrect relations.

To the best of our knowledge, this is the first work that specifically focuses on detecting incorrect `rdfs:subClassOf` relations in large Knowledge Graphs. In contrast with other types of relations, finding such erroneous class subsumption assertions has attracted less attention, possibly due to the fact that the creation of such assertions is rarely automated. Focusing on other types of relations, Ma el al. [2] proposed a method for detecting wrong type assertions by relying on disjointness axioms which they lean using inductive logic programming. Paulheim and Bizer [3] proposed a statistical method for finding erroneous statements for each type of relation by identifying edges whose subject and object type strongly deviate from the characteristic distribution. The largest amount of work on knowledge graph refinement have focused on detecting erroneous equivalence and identity relations. In this family of approaches, various kinds of information have been exploited to identify erroneous statements, such as the description related to the linked resources [4,5], domain knowledge that is included in the ontology or obtained from experts [6,7], and different network metrics [8,9].

2 Algorithm

The algorithm consists of two parts: data pre-processing as in Section 2.1 and the automated refinement as in Section 2.2. Implementation details are included in Section 2.3.

2.1 Data Pre-processing

The subgraph G_{cat} is a collection of all the triples in the form (s, `rdfs:subClassOf`, o). For the sake of efficiency, those classes without any subclasses (leaf nodes) are temporally removed, since by definition they can not be part of any cycle. We further remove all reflexive relations since they form trivial self-cycles. Both of these operations can be done in linear time regarding the number of nodes in the graph.

In addition, we make use of existing equivalence and identity relations to make direct decisions regarding certain subsumption relations. Specifically, all `rdfs:subClassOf` statements between s and o are removed if an equivalence relation between s and o exists in the form of `owl:equivalentClass` or `owl:sameAs`, after transitive closure.

2.2 Automated Cycle-resolving

We resolve the cycles iteratively from local neighbourhoods to cross-domain cycles and eventually ensure the entire knowledge graph is cycle-free. This implies that the amount of edges we remove is not minimal but an approximation to it.

In each round, the algorithm obtains a small subgraph within which we detect the cycles, which we then encode in a MAXSAT solver which makes decision about which relations to remove. We repeat this process until there is no cycle in the entire graph, or until processing time runs out. This gives us an anytime algorithm which gives us increasingly better approximations of a cycle-free graph. Algorithm 1 provides an overview of the method in pseudocode.

Retrieve Local Simple Cycles First we retrieve some cycles in G_{cat} (See Section 2.3 for implementation details). We collect the nodes N of these cycles until we reach an upper-bound B. From these nodes N, we obtain the corresponding neighbourhood $G[N]$. Next, we retrieve all the simple cycles within this subgraph.

For a graph, a simple cycle, or an elementary circuit, is a closed path where no node appears twice except that the first and last node are the same. The time complexity for a function to obtain all the simple cycles is $O((n+e)(c+1))$ for n nodes (classes), e relations (edges) and c elementary circuits [10].

Algorithm 1: Refinement of the Subsumption Relations on Classes

Result: A refined knowledge graph with no cyclic subclass assertions
initialization: retrieve subgraph G_{cat} from G;
pre-processing;
while G_{cat} *is not cycle free and not timeout* **do**
  obtain a set of nodes N with soft upperbound B on size from G_{cat};
  subgraph $G[N]$ corresponding to N;
  generate *simple cycles* SC from $G[N]$;
  encode SC to a MAXSAT solver and obtain a model m;
  decode the results from m and remove relations from G_{cat} accordingly.
end
Export all the removed edges E;
return G_{cat}

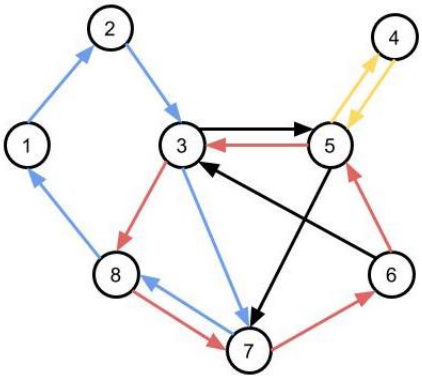


Fig. 1: An example graph

Consider the local neighbourhood in Fig. 1, over nodes $\{1, 2, \ldots, 8\}$, with asserted cycles indicated in blue, yellow and red. We generate all the simple cycles: $1 \to 2 \to 3 \to 5 \to 7 \to 8 \to 1$, $1 \to 2 \to 3 \to 8 \to 1$, $1 \to 2 \to 3 \to 7 \to 8 \to 1$, $3 \to 5 \to 7 \to 6 \to 3$, $3 \to 5 \to 3$, $3 \to 8 \to 7 \to 6 \to 3$, $3 \to 8 \to 7 \to 6 \to 5 \to 3$, $3 \to 7 \to 6 \to 3$, $3 \to 7 \to 6 \to 5 \to 3$, $4 \to 5 \to 4$, $8 \to 7 \to 8$, and $5 \to 7 \to 6 \to 5$. A non-simple cycle would be $5 \to 7 \to 6 \to 5 \to 3 \to 8 \to 7 \to 5$.

It is obvious that if all simple cycles in a graph are resolved, there will be no cycle in the graph anymore[1]. We therefore list all the simple cycles SC of the corresponding subgraph $G[N]$. Next, we employ a MAXSAT solver to find the smallest number of edges that can be removed to resolve all cycles.

Resolving Simple Cycles Using MAXSAT In each iteration (i.e. for each local neighbourhood, as in Figure 1), we obtain a set of simple cycles SC from

[1] By *resolving a cycle*, we mean that at least one of its edges is removed.

the subgraph $G[N]$ as described above. Next, in order to remove a minor amount of edges to break these cycles, we employ a MAXSAT solver and encode all the cycles to it.

First, we introduce some basic concepts of propositional logic and the definition of a MAXSAT problem for the purpose of our task of cycle resolving. For a directed graph $G = \langle E, V \rangle$, if there is an edge from node (a.k.a. vertices) from v to w ($v, w \in V$), then we have $(v, w) \in E$. A *propositional variable* $p_{v,w}$ represents if there is a directed edge from v to w. If $p_{v,w}$ is True, there is an edge, otherwise not. An assignment is to associate all the propositional variables with the value True or False.

In Figure 1, there are four nodes: v_1, v_2, v_3, v_7 and v_8 in the blue cycle. To break this cycle, we need to remove at least one edge (relation). That is, we need to set at least one of $p_{1,2}$, $p_{2,3}$, $p_{3,7}$, $p_{7,8}$, $p_{8,1}$ to False. Equivalently, we need to make the *clause* $s = \neg p_{1,2} \vee \neg p_{2,3} \vee \neg p_{3,7} \vee \neg p_{7,8} \vee \neg p_{8,1}$ evaluate to True.

In this manner, for each cycle $c_i \in SC$, we generate a clause s_i. To resolve all the cycles, we simply need to find an assignment so that all the clauses evaluate to True. It is easy to notice that an easy solution is to remove all the edges (i.e. simply set all propositional variables to false). This amounts to removing all edges and thus removing all cycles. To maintain as much information as possible, we shall formulate this problem as a Partially Weighted MAXSAT problem.

A *Partially Weighted MAXSAT* problem has constraints in two forms: *soft constraints* and *hard constraints*. The fact that we need to resolve all the cyclic connections corresponds to hard constraints (as encoded above). A soft constraint is in the same form except a weight w_i associated with each clause, denoted (c_i, w_i). The goal of this kind problem is to satisfy all the hard constraints while maximising the sum of the weights associated with the satisfied soft constraints, and thus it is a constrained optimisation problem.

In this case, our soft constraints are simply each of the propositional variables corresponding to the edges (relations) with a fixed identical weight. In this way, the MAXSAT procedure will remove all cycles (i.e. satisfying the hard constraints), while keeping as many of the relations as possible (i.e. maximally satisfying the soft constraints). Without confusion, in this paper, we shall use the term MAXSAT instead of Partially Weighted MAXSAT. For the example above, the hard constraint is s as encoded above and the corresponding soft constraints are $(p_{1,4}, 1), (p_{3,1}, 1), (p_{2,1}, 1), (p_{4,2}, 1)$ (with identical weights of 1 each).

The problem of weighted MAXSAT is Δ_2^P-complete (a linear number of calls on instance size to a SAT oracle). When weights are equal, it is $\Delta^{P_2}[\log n]$-complete (a logarithmic number of calls to a SAT oracle). Our algorithm takes all the weights identically as 1 and we call the MAXSAT solver c times, making this algorithm $c\Delta^{P_2}[\log n]$-complete, where c is the number of simple cycles. Although MAXSAT problems are intractable in general, very good solvers exist that can solve problems of realistic size very efficiently.

2.3 Implementation

The SUBMASSIVE system was implemented in Python and takes advantage of both the `networkx` Python package[2] and the Python binding of Z3 SMT solver[3], together with interaction with the LOD-a-lot[4] HDT file though PyHDT[5]. Despite other large knowledge graphs, the results of this paper is solely about the LOD-a-lot knowledge graph [11]. There are in total 28 billion triples in it, making it one of the largest knowledge graphs. The corresponding subgraph G_{cat} consists of over 4 million entries. It turned out to have many cycles nested inside each other. As a result, it was impossible to retrieve all the cycles due to combinatorial explosion.

Such a complex integration resulted in lack of efficiency in memory use. We therefore wrote a separate script to obtain the edges removed in comparison against the equivalent classes and the SameAs dataset[6]. After comparison, we identified 755 relations as unnecessary and removed them. These relations together with its decisions are loaded to the system.

At each iteration, the `find_cycle` method used an optional list of classes as source S and performed depth-first search. If no cycle was found starting from S, a class was arbitrarily chosen and repeatedly searched until there was a cycle or ended up with an exception of no cycle. To avoid retrieving the same cycle, we removed one random relation of the cycle retrieved from a copy of G_{cat} and continue to the next iteration with this new subgraph G'_{cat}.

It worth noting that a rare case is that only one cycle was found whose size exceeded the soft size bound B. In such case, a random relation would be removed from this cycle. An update of the algorithm was to limit the minimum amount of cycles to be 3 (unless no more cycles found).

Using SUBMASSIVE, a user can easily retrieve all the superclasses of any class (transitive closure) without worries of cyclic cases. Moreover, the system was designed to advantage of both queries about knowledge graphs and calls to functions of packages of graph/network theory. This design gives users the ability to conduct hybrid reasoning. Finally, a modular design improves the flexibility in use.

In addition, we resolve the cycles in the graph of `rdfs:subPropertyOf` and publish the resulting hierarchy.

3 Evaluation

As for evaluation, we first present an analysis of the fully automated version and then compare our results against a semi-automatic version.

The best run from our experiment was when the soft bound $B = 60$, the result has 330 relations removed. Figure 2 illustrates how the total amount of removed

[2] `https://networkx.github.io/`
[3] `https://github.com/Z3Prover/z3`
[4] `http://lod-a-lot.lod.labs.vu.nl/`
[5] `https://github.com/Callidon/pyHDT`
[6] We used the dataset (0-99) published at `https://www.sameas.cc/`.

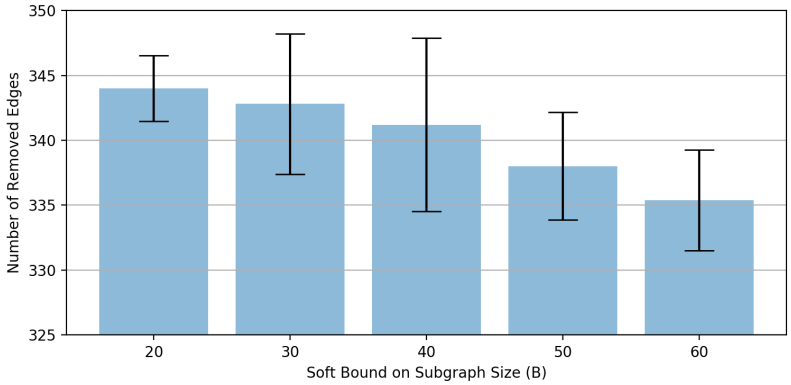


Fig. 2: Number of removed edges against the soft bound on size of subgraph at each iteration (error-bars represent the standard deviation).

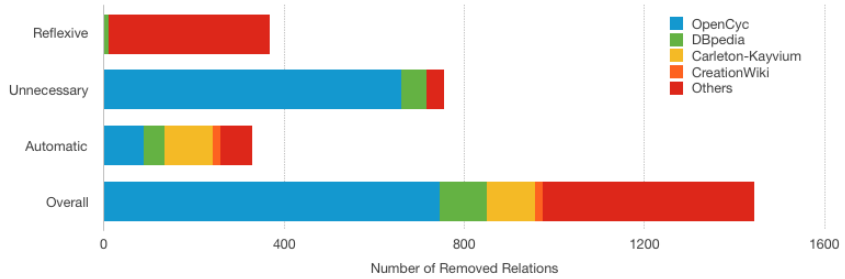
relations decreases as we increase the size B. For each value of B between 20 and 60, we performed the experiment 5 times.[7] When $B \geq 70$, the system faced combinatorial explosions for some runs: there can be over 1 million simple cycles and one iteration may take over one hour or even longer. The results are therefore not included. The variation of removed relations is due to the random removal of edges when obtaining subgraphs.

Fig. 3a shows the sources of the removed relations in stacked bar charts. After comparison against the equivalent classes and the SameAs dataset, we identified 38 and 717 relations respectively, summing up to a total of 755 relations. A closer examination unveils that 87.2% of the unnecessary relations are from the OpenCyc package. It is clear that such cycles were mostly inherited from the OpenCyc, DBpedia and a knowledge base by Kayvium.[8] The source of error could trace back to when these knowledge graphs were constructed from structured/semi-structured knowledge or extracted from text [1]. In addition, a few interdisciplinary relations were removed (less than 12, the exact number depends on the run).

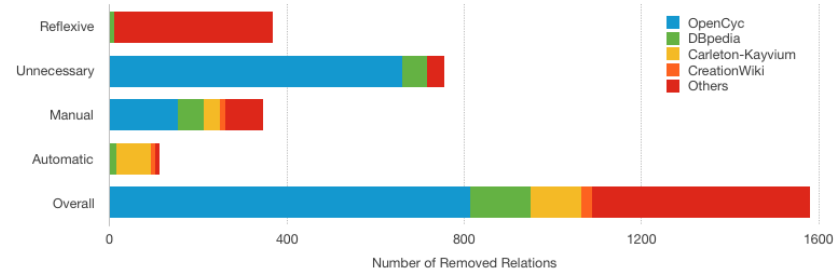
Although it was hard to manually examine each removed relations in its context, it was possible to check pairs of relations forming size-two cycles. We performed an additional manual examination on the relation of these classes. There were a total of 229 size-two cycles, making 458 relations. After the manual

[7] All our experiments were conducted on the LOD Labs machine. It has 32 64-bit Intel Xeon CPUs (E5-2630 v3 @ 2.40GHz) with a RAM of 264GB. The processing time varies between 26 and 99 minutes. We set our time limit to 2 hours, which limits the size of subgraph N to 70.

[8] We use carleton-kayvium as an abbreviation for `http://http-server.carleton.ca/~rgarigue/ontologies/www.kayvium.com/Regional_registry`. The author passed away, leaving the knowledge graphcurrently unmaintained.



(a) Fully automatic (FA)



(b) Semi-automatic (SA)

Fig. 3: An analysis of removed relations in stacked bar-chart.

pre-processing,[9] there were in total 345 relations labelled removed to the best of our knowledge (Fig. 3b). Similarly, we carried out the automatic processing step. The best run from our experiment was when the soft bound $B = 60$, the result has 131 relations removed during the automatic processing step.

We compare the removed relations in two approaches: fully automatic (FA) v.s. semi-automatic (SA). Recall that there were 330 relations removed in FA against 345 relations removed in the manual processing and 131 automatic processing of SA. Out of the 330 relations removed during FA, 165 of them were also removed in the manual process of SA (50.0%) and 57 of them (17.27%) in the automatic process of SA. These two sets summed up to 222 triples (67.27%). A possible reason that the precision was not higher is that the weights on relations for size-two cycles were identical. There was an equal chance for a relation to be removed unless it was nested in other cycles. This calls a better consideration of the context including labels, comments, and relations with other entities. More-

[9] We used the URI and used labels, comments and other information online for decision making. This manual processing removed relations more strictly than the automatic processing which solely aims at resolving cycles. If $A \equiv B$, we consider both relations *unnecessary* and remove them. If unknown, both relations remained.

over, no entry from the FA results were to be removed in the SA process, which showed that our algorithm has a low false-positive (false alarm) rate. In addition, out of the 330 relations removed, 28 appeared to be labelled as 'unknown' in the manual decision process. This result confirmed that the automatic process was considerably reliable.

4 Discussion and Future Work

An immediate benefit of the DAG property is that we are now able to obtain a reliable subclass transitive closure of an entity. This provides the data to perform large-scale study the accuracy of transitive relations regarding different knowledge graph embeddings. In other words, when evaluating embedding-based techniques, if an entity e is of type A, and A is a subclass of B. How likely e is of type B (or its super classes).

We noticed that unnecessary connections came mostly from the OpenCyc knowledge base (87.2%). This might have something to do with how it was created. While this projects treats a class subsumption relations as unnecessary relations when there exists an owl:equivalentClass relation, a future work could be to replace the `owl:equivalentClass` relations by two subsumption relations. Another possible future direction is to convert all the class concepts in `owl:sameAs` relation to one uniform class node, while maintaining the relations with other classes. There could be more relations worth removing.

LOD-a-lot was built from a crawl of the LOD Laundromat in 2015 [11]. Despite that there are many relations removed from the OpenCyc and carleton-kayvium knowledge bases, in the most recent version of DBpedia (2019), there is no cycle found. In contrast to other removed relations, those from the Creation-Wiki come with a biblical worldview. While we did not find any contradictions that were directly due to this difference in worldview, the example does highlight that in a large, heterogeneous knowledge graph, multiple contradicting world-views may be present.

A bottleneck of our algorithm is that we enumerate all simple cycles for encoding. The subgraph G_{cat} of LOD-a-lot turns out to have many cycles nested inside each other. Due to combinatorial explosion, we are only able to perform the algorithm less than one million simple cycles (timeout otherwise). This ensures the subgraph $G[N]$ at the iteration to be cycle-free but limits the size of subgraph N to 60. This implies that the amount of edges we remove is not minimal but an approximation to it. Another future work would be to retrieve all the nodes involved in cycles and then compute a limited amount of simple cycles of the corresponding subgraph at each iteration until all the cycles are resolved.

An alternative way to perform evaluation could be to add or remove some subclass relations from a verified knowledge graph and then use our methods to identify them.

Finally, our work can also serve as a reference for the evaluation the quality of knowledge bases. For example, from our experiment, the carleton-kayviu and creationwiki knowledge bases have serious problems in their ontology while OpenCyc may need a revisit on the use of subclass and equivalent class relations.

5 Conclusion

We have presented an algorithm for the refinement of subclass relations in very large knowledge graphs. Our algorithm takes a hybrid approach, combining the use of network/graph theory and automated reasoning. The fully automatic version does not use external knowledge but still generates relatively reliable results. With some additional manual processing we show that these results can be considerably improved without sacrificing scalability. We have applied our method to the LOD-a-lot dataset. The results—two versions of the cycle-free subclass hierarchy of the LOD-a-lot together with the subProperty hierarchy—are available at `www.submassive.cc`.

Acknowledgements The project is a part of the MaestroGraph project funded by the NWO TOP grant. We thank Luigi Asprino, Wouter Beek and Jacopo Urbani for their helpful advice.

References

1. Heiko Paulheim. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic web*, 8(3):489–508, 2017.
2. Yanfang Ma, Huan Gao, Tianxing Wu, and Guilin Qi. Learning disjointness axioms with association rule mining and its application to inconsistency detection of linked data. In *Chinese Semantic Web and Web Science Conference*, pages 29–41. Springer, 2014.
3. Heiko Paulheim and Christian Bizer. Improving the quality of linked data using statistical distributions. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 10(2):63–86, 2014.
4. Heiko Paulheim. Identifying wrong links between datasets by multi-dimensional outlier detection. In *WoDOOM*, pages 27–38, 2014.
5. John Cuzzola, Ebrahim Bagheri, and Jelena Jovanovic. Filtering inaccurate entity co-references on the linked open data. In *International DEXA Conference*, pages 128–143. Springer, 2015.
6. Aidan Hogan, Antoine Zimmermann, Jürgen Umbrich, Axel Polleres, and Stefan Decker. Scalable and distributed methods for entity matching, consolidation and disambiguation over linked data corpora. *Web Semantics: Science, Services and Agents on the World Wide Web*, 10:76–110, 2012.
7. Laura Papaleo, Nathalie Pernelle, Fatiha Saïs, and Cyril Dumont. Logical detection of invalid sameas statements in rdf data. In *International Conference EKAW*, pages 373–384. Springer, 2014.
8. Christophe Guéret, Paul Groth, Claus Stadler, and Jens Lehmann. Assessing linked data mappings using network measures. In *Extended Semantic Web Conference*, pages 87–102. Springer, 2012.
9. Joe Raad, Wouter Beek, Frank van Harmelen, Nathalie Pernelle, and Fatiha Saïs. Detecting erroneous identity links on the web using network metrics. In *International Semantic Web Conference*, pages 391–407. Springer, 2018.
10. Donald B Johnson. Finding all the elementary circuits of a directed graph. *SIAM Journal on Computing*, 4(1):77–84, 1975.
11. Javier David Fernandez Garcia, Wouter Beek, Miguel A Martínez-Prieto, and Mario Arias. LOD-a-lot: A queryable dump of the lod cloud. 2017.