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, 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. In this paper, we present 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. a directed 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. Only when given a cycle-free subclass hierarchy, can we obtain a reliable subclass transitive closure of an entity. It is required for the evaluation of the accuracy of transitive relations regarding different knowledge graph embeddings.

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, otherwise they represent 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, and hence are less prone to error. 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 addition, we make use of existing equivalence and identity relations to make direct decisions regarding certain subsumption relations. Specifically, we remove all `rdfs:subClassOf` statements between s and o (as *unnecessary* relations) when there is an explicit assertion of `owl:equivalentClass` or the transitive closure of `owl:sameAs`.

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 iteration, there are two steps: obtain 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. Bounded by an upper-bound B, we collect a set of nodes by retrieving cycles in G_{cat} (see Section 2.3 for implementation details). 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 subclass subsumption subgraph $G_{cat}$ from $G$;
pre-processing $G_{cat}$;
while $G_{cat}$ is not cycle free and not timeout do
    obtain a set of nodes $N$ with soft upperbound $B$ on size from $G_{cat}$;
    obtain neighbourhood 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 $E'$ accordingly and get $G'_{cat}$.
end
export all the removed edges $E'$;
return $G'_{cat}$
```

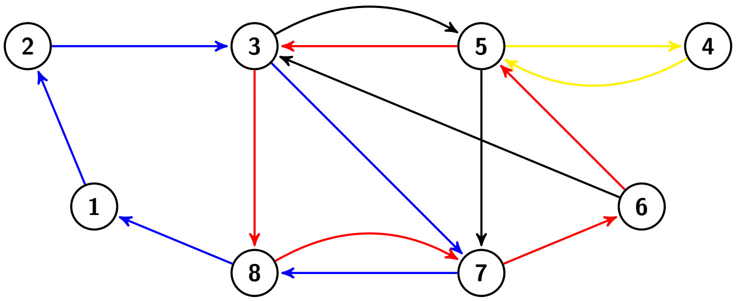


Fig. 1: An example graph

Consider the local neighbourhood in Figure 1, over nodes $\{1, 2, \ldots, 8\}$, with asserted cycles indicated in blue, yellow and red. The simple cycles are $1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 7 \rightarrow 8 \rightarrow 1$, $1 \rightarrow 2 \rightarrow 3 \rightarrow 8 \rightarrow 1$, $1 \rightarrow 2 \rightarrow 3 \rightarrow 7 \rightarrow 8 \rightarrow 1$, $3 \rightarrow 5 \rightarrow 7 \rightarrow 6 \rightarrow 3$, $3 \rightarrow 5 \rightarrow 3$, $3 \rightarrow 8 \rightarrow 7 \rightarrow 6 \rightarrow 3$, $3 \rightarrow 8 \rightarrow 7 \rightarrow 6 \rightarrow 5 \rightarrow 3$, $3 \rightarrow 7 \rightarrow 6 \rightarrow 3$, $3 \rightarrow 7 \rightarrow 6 \rightarrow 5 \rightarrow 3$, $4 \rightarrow 5 \rightarrow 4$, $8 \rightarrow 7 \rightarrow 8$, and $5 \rightarrow 7 \rightarrow 6 \rightarrow 5$. A non-simple cycle is $5 \rightarrow 7 \rightarrow 6 \rightarrow 5 \rightarrow 3 \rightarrow 8 \rightarrow 7 \rightarrow 5$.

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

First, we introduce propositional logic and the definition of a MAXSAT problem for cycle resolving. For a directed graph $G = \langle E, V \rangle$, if there is an edge from

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

a node 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 False then we remove the edge. An assignment is to associate all the variables with True or False.

In Figure 1, we can find the following five nodes in the blue cycle: v_1, v_2, v_3, v_7 and v_8. To resolve it, we need to remove at least one edge. Hence, we set at least one of the following variables $p_{1,2}$, $p_{2,3}$, $p_{3,7}$, $p_{7,8}$, $p_{8,1}$ 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 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 type of problems 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). For fairness, we assign a fixed identical weight each. 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). For simplicity, we use in the term MAXSAT in reference to Partially Weighted MAXSAT. For the example above, the hard constraint is s as encoded above and the corresponding soft constraints are propositional variables 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 and the number we call the MAXSAT solver i is bounded by the number of nodes n. To handle combinatorial explosion, we introduce a bound B to limit the number of simple cycles at each iteration. Although both cycle-finding and cycle-resolving in this setting are intractable, efficient solvers/programs exist that can solve problems of realistic size.

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 using the PyHDT li-

[2] `https://networkx.github.io/`

[3] `https://github.com/Z3Prover/z3`

[4] `http://lod-a-lot.lod.labs.vu.nl/`

brary.[5] The LOD-a-lot dataset represents the graph merge of 650K datasets crawled from the LOD Cloud in 2015 [11]. It contains over 28 billion triples, making it one of the largest publicly available knowledge graphs. The corresponding subgraph G_{cat} has over 4 million edges. There are several cycles nested inside each other, which results in 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.cc 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 uses an optional list of classes as source S and performs a depth-first search. If no cycles are found starting from S, a class is 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}.

Our simple cycles are retrieved using the `simple_cycle` method. The algorithm uses a trick to block duplicated search from a root and then build the elementary path from the searching results. Details are in [10].

It is 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 3 (unless no more cycles found).

Using SUBMASSIVE,[7] a user can retrieve, for any given class, all its superclasses (i.e. its transitive closure) without worries of cyclic cases. Using a variant of the system, we resolve the cycles in the graph of `rdfs:subPropertyOf` and publish the resulting hierarchy at the same place in N-Triples format. The size of the resulting cycle-free datasets is 829.6MB for subClassOf, and 14.6MB for subPropertyOf.

3 Evaluation

We present our evaluation of the fully automated version (Section 3.1) and then compare it against a semi-automatic version (Section 3.2).

3.1 Evaluation of the Fully Automated Version

Figure 2 illustrates how the total amount of removed relations decreases as we increase the size B. The blue bars indicates the average and the error-bars represent the standard deviation. The best run from our experiment has 330 relations were removed as a result (soft bound $B = 60$). The algorithm is not deterministic because we remove a random edge for each cycle when retrieving the neighbourhood. For each value of B between 20 and 60, we performed the experiment 5

[5] `https://github.com/Callidon/pyHDT`

[6] We use the dataset (closure_099) published at `https://zenodo.org/record/3345674`

[7] The code is opensource at `www.submassive.cc`

times to obtain the mean and variance (since the algorithm is not deterministic).[8] 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 (as illustrated in Figure 2 as error bars) of the removed relations is due to the random removal of edges when obtaining subgraphs. The figure shows a decrease in number of edges removed as the bound enlarges.

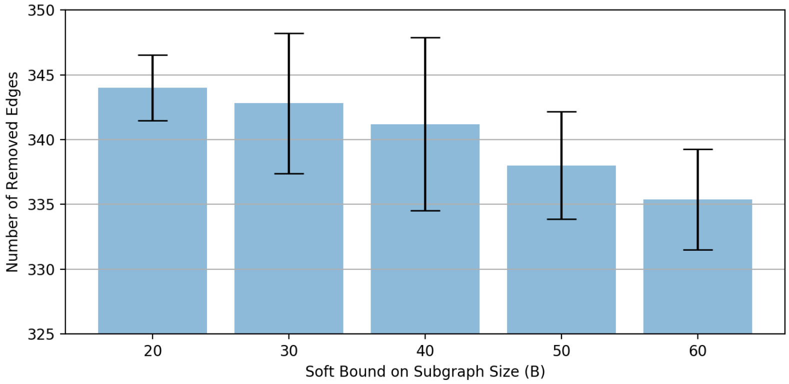


Fig. 2: Number of removed edges against the soft bound on size of subgraph at each iteration (error-bars represent the standard deviation).

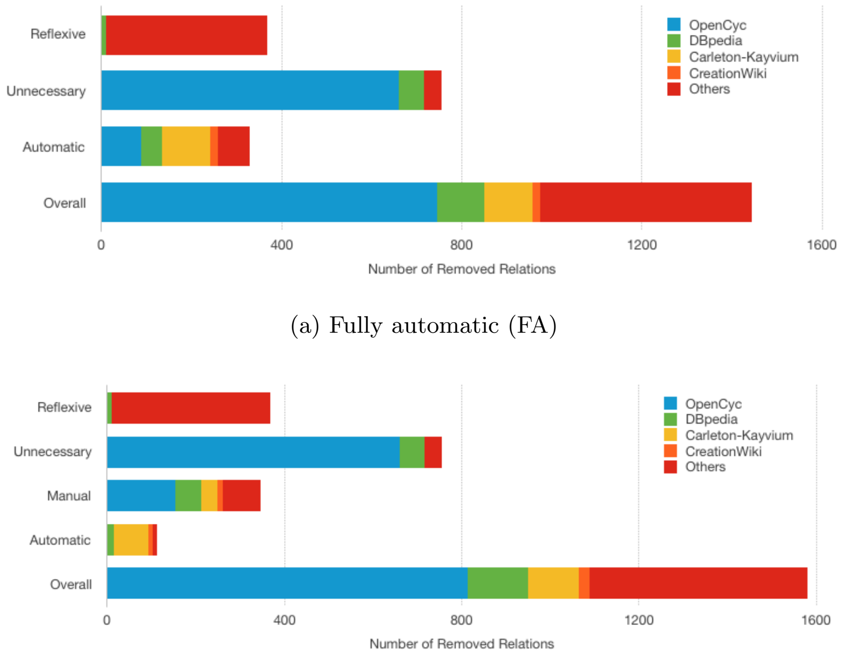
Figure 3a shows the sources of the removed relations. After comparison against the equivalent classes and the sameAs.cc dataset, we identified 38 and 717 relations respectively, summing up to a total of 755 relations. A closer examination unveils that 87.2% of the unnecessary relations have a subject with the OpenCyc namespace. It is clear that such cycles were mostly inherited from the OpenCyc, DBpedia and a knowledge base by Kayvium.[9] 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).

3.2 Evaluation against the Semi-automated Version

Since there is no gold standard available, we manually checked the correctness of relations resulting in size-two cycles. There were a total of 229 size-two cycles,

[8] All our experiments were conducted on the following machine: 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.

[9] We use carleton-kayvium as an abbreviation for `http://http-server.carleton.ca/~rgarigue/ontologies/www.kayvium.com/Regional_registry`. The knowledge graph suffers heavily from erroneous subclass relations.



(a) Fully automatic (FA)

(b) Semi-automatic (SA)

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

forming 458 relations. After the manual pre-processing,[10] there were in total 345 relations identified as erroneous to the best of our knowledge (Figure 3b). Following that, we conduct the automatic part. The best run has 131 relations removed ($B = 60$).

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 reason that the precision was not higher is that the weights on relations for size-two cycles were identical when resolving automatically. Thus, there was an equal chance for a relation to be removed unless it was nested in other cycles in

[10] We relied on the IRI, 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 remain.

the iteration. Moreover, no entry from the FA results were to be removed in the SA process, which indicates that our result has a low false-positive rate.

4 Discussion

We have presented an algorithm for the refinement of subclass relations in very large knowledge graphs and also compared against its semi-automatic version. 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 global minimal but an approximation to it. Another alternative approach 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 cycle-free.

LOD-a-lot was built from a crawl of the LOD Laundromat in 2015 [11]. In the most recent version of DBpedia (2019), there is no cycle found. In contrast to other removed relations, those from the creationwiki (`http://creationwiki.org`) come with a biblical worldview. Despite its cyclic assertions, we did not find any contradictions that were directly due to this difference in worldview. This highlights that in LOD-a-lot, multiple contradicting worldviews may be present.

Although there are cycles involving cross-domain relations, most removed relations are within the same domain. An explanation is that since relations within the domain are also involved in other cycles, for the sake of optimisation, they are removed. Another possible reason is that our approach prioritises the resolving of local cycles. Most removed ones are between domains `http://www.daml.org`, `http://ontology.ihmc.us`, and `http://www.w3.org`.

Considering there is no ground truth in very large knowledge graphs, our evaluation has its limits. 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.

5 Conclusion and Future Work

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 tested our approach on the LOD-a-lot dataset. The resulting cycle-free subclass hierarchy and the source code are available at `www.submassive.cc`.

While this work considers a class subsumption relation as unnecessary when there also 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.

Our work also examines the quality of knowledge graphs. For example, the carleton-kayviu and creationwiki knowledge graphs have complex subclass cycles in their ontology while OpenCyc needs a revisit on the use of subclass and equivalent class relations.

Finally, with this cycle-free graph, we are now able to obtain a reliable subclass transitive closure of an entity. This provides the data to perform Machine Learning on large-scale, such as evaluating the accuracy of transitive relations regarding different knowledge graph embeddings. In other words, if an entity e is of type A, and A is a subclass of B, we would like to know how likely e is also of type B (or its super classes, without having to deal with incorrect cycles).

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 and anonymous reviewers 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.