Discovering Semantically Broken Links in LOD Datasets

André Gomes Regino[1][0000−0001−9814−1482], Julio Cesar dos Reis[1,2][0000−0002−9545−2098]

[1] Institute of Computing, University of Campinas, Campinas - SP, Brazil
andre.regino@students.ic.unicamp.br, jreis@ic.unicamp.br
[2] Nucleus of Informatics Applied to Education, University of Campinas, Campinas - SP, Brazil

Abstract. Links between data elements described by RDF models stand to the core of the Linked Open Data (LOD). Usually, semi-automatic algorithms are used to build connections among RDF datasets. However, RDF assertions are subject to change, which can affect existing links. Interconnected open data demands (semi-)automatic methods and tools to maintain their consistency over time. In this context, an example of inconsistency is the presence of semantically broken links. Their detection remains a very hard task to perform, given the difficulties to interpret the meaning of resources involved in the link. In this paper, we investigate a technique for the detection of semantically broken links between resources in distinct RDF datasets via the use of similarity values and triple changes computed. This study paves the way for the development of (semi-)automatic mechanisms for link maintenance in LOD.

Keywords: Web of Data Evolution; Link Evolution; Semantic Broken Link; Link Changes; Broken Link Detection

1 Introduction

Linked Open Data (LOD) has grown as a movement for structuring and sharing data on the Web. One of the key principles refers to the interconnection between the resources of different datasets in LOD. The acceptance and usage of such good practice by the datasets contribute to the growth and consequently to the success of a new Web of Data, with consistent and robust interlinked datasets representing knowledge of diversified domains.

The interconnection of Resource Description Framework (RDF) statements via explicit links plays a central role to assure data linkage, semantic interoperability, and knowledge discovery. RDF triples - including links - must be updated, added, or removed to keep the repositories up-to-date. The update of linked data is necessary due to the evolutionary characteristic of these structured datasets, following the evolution of the knowledge they represent. However, data changing operations might influence well-formed links, which turns the maintenance of their consistencies over time a hard task. Given their evolutionary characteristics, the datasets face a big and well-known challenge: the integrity of links.

An open problem associated with this challenge is the presence of broken links. One type of inconsistency found in the literature is the semantically broken link. It is found

at instances of links between two different datasets that evolved in a given period of time. The literature has addressed cases of structurally broken links because detecting that a part of the link was removed is easier than detecting changes in the meaning of the resources. Nevertheless, cases where there is a change in a resource of the link (subject or object) needs to be further investigated. Literature has addressed techniques to track changes in RDF repositories. However, the task of maintaining the links up-to-date requires deeper studies. Few investigations approached the link integrity problem aiming to monitor and preserve data quality [7].

A misspelling in the resource of a link can be easily checked by traditional algorithms of ontology management. However, a change in the semantics of a resource is not that simple for automatic semantic inconsistency detection. To illustrate the difficulties in this task, suppose there is a link connecting two resources labeled as "São Paulo" in two different datasets. If there is a change in the subject of the link from "São Paulo" to "City of São Paulo", we have to guarantee that the link connecting this resource to the object remains consistent in the second dataset, and its semantics is preserved. If the linked resource is connected by the predicate "sameAs", or any other equality predicate, and the object of the connection possesses the same meaning of "City of São Paulo", then the link is not corrupted. However, the object of the connection can be, for example, "Greater São Paulo", which is different from "City São Paulo". This is an example of a semantically broken link.

In this article, we propose a novel methodology to detect semantically broken links in RDF datasets. The algorithm uses as input two versions of the same dataset at different periods of time. On this basis, our solution detects instances of semantically broken links. The methodology explores the results of syntactic and semantic similarity measures that calculate the degree of proximity between the resources that changed from one dataset release to another.

The remaining of this paper is organized as follows: Section 2 discusses related work; Section 3 presents our proposed methodology; Section 4 reports on the conducted experimental evaluation; Sections 5 discusses the obtained findings whereas Section 6 shows the final remarks of this paper.

2 Related Work

One of the first efforts in the topic of broken links came from DSNotify [2], a tool that is able to recognize structurally broken links and fix them, supervised by a specialist. DSNotify sends notifications to the maintainer of the datasets warning that something changed in the dataset. DSNotify served as an inspiration to the subsequent alternatives found in literature, like Delta-LD [12], which is able to fix broken links removing the inconsistent resource, and reconnecting the link using SPARQL templates. However, both alternatives only deal with structurally broken links. The framework Silk [13] was developed to maintain links between ontologies. It generates new links between datasets, evaluates them, and track possible new links that can be used in both datasets. However, it does not check the consistency of existing links.

It is relevant to differentiate structurally and semantically broken links. A link is structurally broken, as stated by Singh, Brennan and O'Sullivan, "if either source or

target is no longer dereferenceable" [12]; and also by Popitsch and Haslhofer [6] "if its target resource had representations that are not retrievable anymore". A link is semantically broken when the semantic of data in the target dataset is not the same as the semantic of the source [3]. Semantically broken links are harder to detect and fix than structurally [6].

Pruski *et al.* [8] investigated the semantic evolution aspect of mappings, which are the links in the conceptual level (ontologies). They explored background knowledge on their approach. This work differs from ours because it does not deal with LOD datasets at the instance level.

In our recent studies, we investigated if there is a correlation between changes in triples and changes in links [11]. Changes in triples associated with the link can be a clue to identify cases of broken links. Regino *et al.* [10] studied a thorough literature survey concerning aspects of link maintenance in the LOD context. The authors organized existing approaches into two groups: preservation solutions, related to the identification and notification of broken links; and maintenance solutions, related to the edition and fixing of the broken links. The study found that most solutions fall into the preservation solutions instead of maintenance, which still requires research efforts to fully address the problem.

Although literature presents contributions to the management and analysis of links in RDF datasets, to the best of our knowledge, it still lacks solutions including algorithm, framework, and experimental studies - aiming to fully and automatically address the problem of semantic broken link in LOD context.

3 Identification of Semantically Broken Links

A RDF triple refers to a data entity composed of subject, predicate, and object defined as $t = (s, p, o)$. A dataset in LOD is a conglomeration of a finite number of RDF triples in a domain. Formally, $\mathcal{R} = \{t_1, t_2, t_3, ..., t_n\}$. Besides the use of triples inside a dataset, the linkage among several datasets interconnect the datasets. To this end, it is necessary that a predicate establishes a relation between a subject of the first dataset (source) and an object of the second (target). Formally, we define a link as $l =< r_a, p, r_b >$ connecting a pair of resources r_a and r_b, in which $r_a \in \mathcal{R}^S$ and $r_b \in \mathcal{R}^T$, such that $\mathcal{R}^S$ differs from $\mathcal{R}^T$. For the definition of p, we consider predicates $owl : SameAs$. We define a set of links between $\mathcal{R}^S$ and $\mathcal{R}^T$ as follows: $\mathcal{L} = \{l_0, l_1, l_2, ..., l_n\}$. A complete dataset refers to the union of internal triples and links, such as $\mathcal{D} = \mathcal{R} \cup \mathcal{L}$.

The evolution of RDF datasets in terms of changes affecting their triples may invalidate previously determined links. This hampers data linkage consistency over time. In order to maintain the consistency of RDF datasets, its links should remain in an integrity state, even with underlying changes in data.

At this stage, we introduce the notion of time $j \in N$. For a RDF dataset, we denote $\mathcal{D}^{S^j}$ the initial version of the dataset and $\mathcal{D}^{S^{j+1}}$ its evolved version. Consider a link l^j at time j and a link l^{j+1} based on distinct releases of the associated datasets. Modifications occurring in a resource of the link (r_a) from a release of the dataset in a specific time j to a time $j + 1$ can invalidate such link $l^j(r_a \rightarrow r_b)$. For example, r_a can exist at time

j ($r_a \in \mathcal{D}^{S^j}$), but be removed at time $j+1$ ($r_a \notin \mathcal{D}^{S^{j+1}}$). In this sense, the link can be considered structurally broken or semantically invalid.

We investigate the evolution of the subject of a link (note that we are assuming the object resource in the link keeps stable). Given a link $l^j(r_a \rightarrow r_b)$ at time j this link evolved, changing its subject in dataset $\mathcal{D}^S$ from r_a to r_c. This results in $l^{j+1}(r_c \rightarrow r_b)$. In our context, the change in the subject updated its meaning, which results in r_c being semantically different from r_a. The link established at time $j+1$ between $r_c \in \mathcal{D}^S$ and $r_b \in D^T$ can became semantically broken. The update of the link $l^j(r_a \rightarrow r_b)$ may not negatively affect the established link because it can be an update to fix the given link (e.g., a re-connection to a better resource). We address the challenge that given a link, our solution detects whether it is semantically broken based on an analysis of similarity values among involved resources.

Figure 1 presents an real example of links (affected by changes overtime). $\mathcal{D}^{S^j}$ represents DBpedia before evolution, $\mathcal{D}^{S^j+1}$ stands for DBpedia after evolution and $\mathcal{D}^{T^j}$ Wikidata. We only consider evolution from one of the datasets ($\mathcal{D}^S$). Figure 2 presents our methodology to detect the inconsistencies in links.

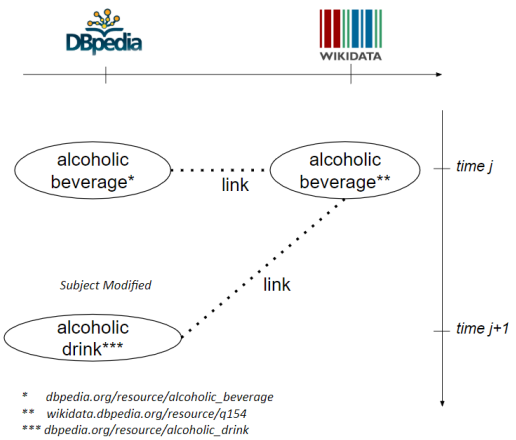

Fig. 1. Example of a Changed Link Connecting DBpedia and Wikidata

Steps 1-2: Reading the Datasets and Link Identification. The required input refers to the same dataset in two distinct versions $\mathcal{D}^{S^j}$ and $\mathcal{D}^{S^{j+1}}$. Both versions are composed of RDF triples and links connecting to an external dataset $\mathcal{D}^T$. The second step is the separation of links from other triples. First, we discover which predicates in the dataset represent the connection between subject and objects with different datasets. This is performed by examining the host of both subject and object. If they are different, then the subject and object belong to different datasets, which classifies this triple as a link. Each triple that contains this predicate is then considered a link. Afterward, the solution retrieves all triples that have the predicates categorized as links. At the end of this step, we have two lists of links: $\mathcal{L}^j$ (regarding $\mathcal{D}^{S^j}$) and $\mathcal{L}^{j+1}$ ($\mathcal{D}^{S^{j+1}}$).

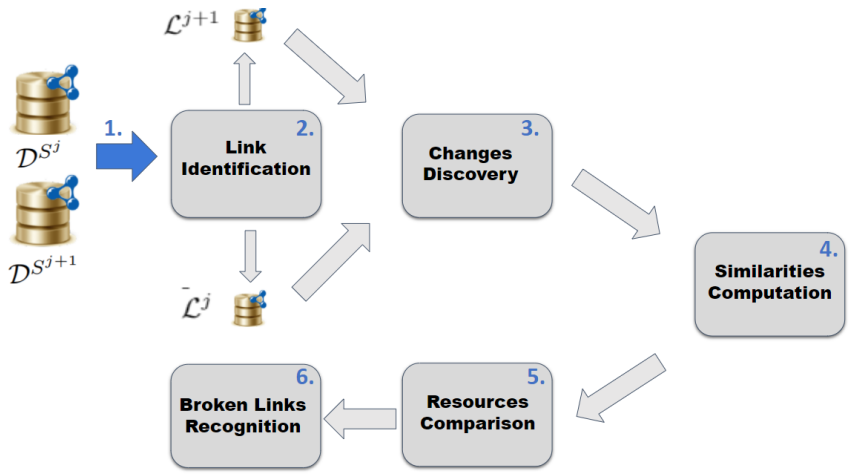

Fig. 2. Methodology to Discover Semantically Broken Links

Step 3: Changes Discovery. In the third step, our solution compares each link from $\mathcal{L}^j$ with $\mathcal{L}^{j+1}$, recovering all links that had its subject changed, but maintained its predicate and object. Note that a link also could change its predicate, but this case is not addressed in this study (a subject for future work).

Figure 1 shows two links as an example of an evolved subject of a link. The triples are composed of a subject from *DBpedia*, a predicate "sameAs", and an object from Wikidata. The first triple (upper side) represents the triple before evolution (time j) with the subject "alcoholic beverage". The second line (lower side) represents the triple after evolution (time $j+1$) with the subject "alcoholic drink". It is valid to note that among these changes, there are cases of consistent and inconsistent changes. To determine if these changes break the link, we perform similarity analyses among the affected resources.

Step 4: Similarity Computation. At this step, the solution applies a similarity computation to measure the degree of relatedness between the resources. The input is composed of two URIs. The URI is mainly composed of 2 pieces: the host and the path. The host represents the dataset and the path may represent the resource. The path in some datasets is composed of a sequential identification number. In order to calculate the similarity between the resources, there is a need to retrieve a label associated with this sequential number, otherwise we would compare sequential numbers and labels, which is undesirable. To perform it, the algorithm retrieves the object correspondent to the property *rdfs:label*. The output of this step is a normalized value ranging from 0 to 1, which values closer to 1 represents more similar results.

Step 5: Resources Comparison. As presented in Figure 3, our solution computes three evolution mensurations concerning similarity.

– **Before evolution computation:** Compute similarity value between r_a and r_b at time j. This evaluates what was the similarity between subject and object of the

link **before** the evolution. In the example in Figure 1, we calculate the similarities between subject "alcoholic beverage" from DBpedia and object "alcoholic beverage" from Wikidata at time j;

– **After evolution computation:** Compute similarity value between r_c and r_b at time $j+1$. This evaluates what is the similarity between subject and object of the link **after** the evolution. In the example in Figure 1, we calculate the similarities between "alcoholic drink" from DBpedia and object "alcoholic beverage" from Wikidata at time $j+1$;
– **Subject evolution computation:** Compute the similarity value between r_a at release j and r_c at time $j+1$. In the example in Figure 1, we calculate the similarity between **subjects** "alcoholic beverage" from DBpedia at time j and "alcoholic drink" from DBpedia at time $j+1$.

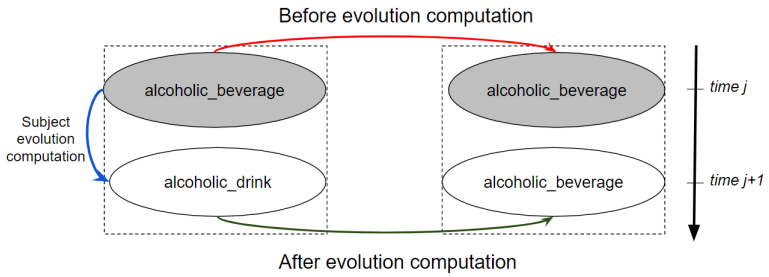

Fig. 3. Mensurations applied in our algorithm

Step 6: Broken Links Recognition. For each set of comparisons between the resources (measures concerning before evolution, after evolution, and subject evolution measures), our solution identifies semantically broken links. Our key assumption relies on the fact that the link is semantically broken if the value of syntactic and semantic similarity before the evolution is higher than after the evolution measure. We understand that in a semantically broken link, the similarity among the involved resources decreased in comparison with the original link at time j. In this sense, the rational to assign a semantically broken link considers that if the "before evolution computation" (similarity value) outperforms both "after evolution computation" and "subject evolution computation", our solution raises a flag of a possible case of a semantically broken link.

Algorithm 1 defines our procedure to discover inconsistent links. After obtaining the links and their changes (lines 3 - 6), the algorithm computes the similarity values. To this end, the algorithm selects the subject in its versions j and $j+1$ and the object of the links (lines 7 - 9). The three defined measures are then calculated (lines 11 - 13) in order to identify and store the broken links (lines 14 - 15). Our work fully implemented our methodology and algorithm[3].

[3] https://gitlab.ic.unicamp.br/jreis/evoLOD-analysis

Algorithm 1 Discovery of Broken Links

Require: $\mathcal{D}^{S^j}, \mathcal{D}^{S^{j+1}}$

$\mathcal{L}^j \leftarrow \emptyset$;
$\mathcal{L}^{j+1} \leftarrow \emptyset$;
$\mathcal{L}^j \leftarrow getLinks(\mathcal{D}^{S^j})$;
$brokenLinks \leftarrow \emptyset$;
$\mathcal{L}^{j+1} \leftarrow getLinks(\mathcal{D}^{S^{j+1}})$;
$\Delta \leftarrow detectChangedLinks(\mathcal{L}^j, \mathcal{L}^{j+1})$;
for all $l \in \Delta$ **do**
  $r_a^j \leftarrow getOldSubject(l)$;
  $r_b^j \leftarrow getOldObject(l)$;
  $r_c^{j+1} \leftarrow getNewSubject(l)$;
  $SimBefEvol \leftarrow calculateSimilarityBeforeEvolution(r_a^j, r_b^j)$;
  $SimAfterEvol \leftarrow calculateSimilarityAfterEvolution(r_c^{j+1}, r_b^{j+1})$;
  $SimBetweenSubj \leftarrow calculateSimilarityOfSubjects(r_a^j, r_c^{j+1})$;
  if $SimAfterEvol \geq SimBefEvol \wedge SimBetweenSubj \geq SimBefEvol$ **then**
    $brokenLinks \leftarrow l$;
  end if
end for
return $brokenLinks$

4 Experimental Evaluation

The goal of this evaluation is to assess our solution for semantically broken link detection based on real-world datasets. In our experiments, we considered DBpedia[4], given its openness characteristic, as long as its constantly updated over time combined with the presence of numerous links connecting to other datasets in LOD. We also chose Wikidata[5] and GeoNames[6] datasets as the income of the links. The characteristic of evolution is important for this evaluation because these datasets are subject to new releases periodically. Our solution is tailored to verify the consistency of the links when a new version of the dataset is released.

We used as input two versions of DBpedia, from October 2015 and October 2016. DBpedia itself has approximately 10 billion triples in total (considering all chapters). In order to speed up the process of reading the datasets, we recovered only links from DBpedia to Wikidata and those from DBpedia to GeoNames that were modified between the versions to reduce the input. We acquired 174 modified links (87 before evolution $\mathcal{L}^j$ and 87 after evolution $\mathcal{L}^{j+1}$) from DBpedia to Wikidata; and 7950 modified links (3975 before evolution $\mathcal{L}^j$ and 3975 after evolution $\mathcal{L}^{j+1}$) from DBpedia do GeoNames.

The only predicate used to link resources from the datasets is the "same as"[7]. Before calculating the similarity values, a request for each object of the link (target dataset) was performed to retrieve the label associated with the resource.

[4] http://dbpedia.org
[5] http://wikidata.org
[6] http://geonames.org
[7] http://www.w3.org/2002/07/owl#sameAs

On the selected part of the dataset, we applied our Algorithm 1. This retrieved the label and calculated the similarities addressing each link. The solution in our experimental study calculated the similarity value considering three distinct similarity techniques.

Syntactic similarity stands for the calculation on how equal are two given character strings in terms of lexical analysis. A semantic similarity computes if two input terms share the same meaning even though they differ in a lexical way. A popular method to calculate the semantic similarity between terms is to find the minimum length of path connecting them based on background knowledge, if the under analysis terms are represented in a hierarchy of concepts [9]. The syntactic similarity was calculated using the Levenshtein Distance [4]. The semantic similarity was calculated using the background knowledge database known as WordNet, a free lexical database organized as a hierarchy of concepts. The similarity value was calculated based on how many nodes are between two terms. The third similarity technique was the Nasari [1], which uses as background knowledge the *BabelNet* semantic network [5].

Following the example in Figure 1, the Levenshtein algorithm resulted in a broken link detection case, once that the first similarity value (between the older subject and object) returned a higher value than the second value (between the newer subject and object). The algorithm which uses WordNet as background knowledge resulted in no semantic change because all three measures returned the same value of semantic similarity. The same occurred to the algorithm that uses BabelNet as background knowledge.

We applied our algorithm several times (one time considering each similarity function investigated). Table 1 presents the number of broken links detected in each dataset when using each different similarity function in our algorithm.

Each pair of 87 links from Wikidata generated 3 results (one for each similarity function). The total number of broken links cases found was 38 (cf. Table 1)(14.55% of the total number of changed links in Wikidata). Applying our algorithm with Levenshtein similarity, we found 33 out of the 38 (86.84%). With the use of WordNet and BabelNet, our algorithm identified 4 and 1 case of broken links, respectively.

An example of a broken link identified by Levenshtein in Wikidata's dataset states that it became broken after the resource "vacation property" (DBpedia) linked to "vacation property" (Wikidata) change to "holiday cottage" (DBpedia) linked to "vacation property" (Wikidata). However, the change is valid, since both resources are synonyms, implying that the detected broken link by Levenshtein is a false positive. WordNet and BabelNet identified it correctly as a not broken link case.

Each pair of 3975 links from GeoNames generated 3 results (one for each similarity). The total number of broken links cases found were 1551 (cf. Table 1)(13% of the total number of changed links in Geonames). Applying our algorithm with Levenshtein similarity, we found 1502 out of 1551 (96.84%). WordNet and BabelNet datasets identified 29 and 20 cases of a broken link, respectively.

An example of a broken link identified by Levenshtein in Geonames's dataset states that it became broken after the resource "niagara falls" (DBpedia) linked to "niagara falls" (GeoNames) changed to "niagara river" (DBpedia) linked to "niagara falls" (GeoNames). This example of a broken link was detected by our algorithm via the use of WordNet and BabelNet as background knowledge. This is a true positive case, since

	Wikidata	Geonames
Levenshtein	33	1502
Wordnet	4	29
Babelnet	1	20
Total	38	1551

Table 1. Identified broken links in the evolution of the studied datasets with our algorithm with the use of distinct similarity measures

"niagara falls" is a component of "niagara river", not a synonym. The change then turns it broken, correctly identified by all 3 applications of our algorithm.

5 Discussion

Handling the impacts of RDF datasets evolution in established links is essential to maintain the consistency and benefits of LOD over time. This is especially valuable because of the heavy interconnection presented in LOD. We proposed an algorithm that discovers cases of semantically broken links. We conceived and developed a methodology that distinguishes the prejudicial and non-prejudicial changes to the links, regarding the changes in the subject of links.

We found semantically broken links in the studied dataset via our solution. There is still room for improvement in the process of updating the resources and links of the dataset. The majority of broken links were detected with the use of the Levenshtein similarity associated with our algorithm. However, we found that Levenshtein is not suitable, since it returns a high number of false positives. On the other hand, the results based on the use of WordNet and BabelNet represent a small number of the total. This might have been caused by the use of background knowledge based on a generic domain. The use of background knowledge from specific domains that match the one used in the datasets or a specialized dictionary would be more suitable to improve the effectiveness.

For future work, we aim to aggregate new measures of similarities. One idea is to explore the neighborhood of the resources and compare them. We can also look at the ontology level, not only at the instances, thus investigating the graph structure.

6 Conclusion

Links are the most valuable assets in LOD. It is necessary to have adequate methods to keep them up-to-date over time. New releases of RDF datasets may lead to inconsistent links. This paper proposed a novel technique to discover cases of semantically broken links using syntactic and semantic similarity measures. This study was the first step to gather techniques to develop a framework responsible for the (semi-)automatic maintenance of links in LOD. Our future work involves the full implementation of this framework.

Acknowledgements

This work is financially supported by the São Paulo Research Foundation (FAPESP) (grants #2017/02325-5, #2018/14199-7 and #2013/08293-7)[8].

References

1. Camacho-Collados, J., Pilehvar, M.T., Navigli, R.: Nasari: a novel approach to a semantically-aware representation of items. In: Association for Computational Linguistics: Human Language Technologies. pp. 567–577 (2015)
2. Haslhofer, B., Popitsch, N.: Dsnotify-detecting and fixing broken links in linked data sets. In: 20th International Workshop on Database and Expert Systems Application. pp. 89–93. IEEE (2009)
3. Kovilakath, V.P., Kumar, S.: Semantic broken link detection using structured tagging scheme. In: Proceedings of the International Conference on Advances in Computing, Communications and Informatics. pp. 16–20. ACM (2012)
4. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady **10**, 707–710 (1966)
5. Navigli, R., Ponzetto, S.P.: Babelnet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network. Artificial Intelligence **193**, 217–250 (2012)
6. Popitsch, N., Haslhofer, B.: Dsnotify: A solution for event detection and link maintenance in dynamic datasets. Web Semantics: Science, Services and Agents on the World Wide Web **9**(3), 266–283 (2011)
7. Pourzaferani, M., Nematbakhsh, M.A.: Repairing broken rdf links in the web of data. Int. J. Web Eng. Technol. **8**(4), 395–411 (2013)
8. Pruski, C., Dos Reis, J.C., Da Silveira, M.: Capturing the relationship between evolving biomedical concepts via background knowledge. In: SWAT4LS (2016)
9. Rada, R., Mili, H., Bicknell, E., Blettner, M.: Development and application of a metric on semantic nets. IEEE transactions on systems, man, and cybernetics **19**(1), 17–30 (1989)
10. Regino, A.G., dos Reis, J.C., Bonacin, R., Morshed, A., Sellis, T.: Link maintenance for integrity in linked open data evolution: Literature survey and open challenges. Semantic Web Journal (accepted for publication) (2020)
11. Regino, A.G., dos Reis, J.C., Matsoui, J., Bonacin, R., Morshed, A., Sellis, T.: Understanding link changes in lod via the evolution of life science datasets. In: Workshop Semantic Web solutions for large-scale biomedical data analytics (SeWeBMeDA) at the 18th International Semantic Web Conference (ISWC'19).
12. Singh, A., Brennan, R., O'Sullivan, D.: Delta-ld: A change detection approach for linked datasets. In: 4th Workshop on Managing the Evolution and Preservation of the Data Web (MEPDaW) (2018)
13. Volz, J., Bizer, C., Gaedke, M., Kobilarov, G.: Discovering and maintaining links on the web of data. In: Int. Semantic Web Conference. pp. 650–665. Springer (2009)

[8] The opinions expressed in here are not necessarily shared by the financial support agency.