DELTA-LD:
A Change Detection Approach for Linked Datasets

Anuj Singh, Rob Brennan and Declan O'Sullivan
ADAPT Centre, School of Computer Science and Statistics, Trinity College Dublin, Ireland
{singh.anuj, rob.brennan, declan.osullivan}@adaptcentre.ie

Abstract. This paper presents DELTA-LD, an approach that detects and classifies the changes between two versions of a linked dataset. It contributes to the state-of-art: *firstly,* by proposing a classification to distinctly identify the resources that have had both their IRIs and representation changed and the resources that have had only their IRI changed; *secondly* by automatically selecting the appropriate resource properties to identify the same resources in different versions of a linked dataset with different IRIs and similar representation. The paper also presents the DELTA-LD change model to represent the detected changes. This model captures the information of both changed resources and triples in linked datasets during its evolution, bridging the gap between resource-centric and triple-centric views of changes. As a result, a single change detection mechanism can support several diverse use cases like interlink maintenance and replica synchronization. The paper, in addition, describes an experiment conducted to examine the accuracy of DELTA-LD in detecting the changes between the person snapshots of DBpedia. The result indicates that the accuracy of DELTA-LD outperforms the state-of-art approaches by up to 4%, in terms of F-measure. It is demonstrated that the proposed classification of changes helped to identify up to *1529 additional updated resources* as compared to the existing classification of resource level changes. By means of a case study, we also demonstrate the automatic repair of broken interlinks using the changes detected by DELTA-LD and represented in DELTA-LD change model, showing how 100% of the broken interlinks were repaired between DBpedia person snapshot 3.7 and Freebase.

Keywords: Change detection, link maintenance, dataset dynamics, linked data

1 INTRODUCTION

Many linked datasets are highly dynamic in nature [1]. For an application consuming a dynamic linked dataset, the dynamic nature of the dataset may result in issues for the application such as broken interlinks or outdated data [1].

These issues are typically encapsulated in the research community using the term "dataset dynamics". Dataset dynamics investigates the approaches that detect changes in linked datasets during their evolution, and the vocabularies to represent the detected changes [2]. The aim is to build: (i) vocabularies to represent the change information, (ii) mechanisms for change detection, and (iii) change propagation

methods [2]. Change detection in linked datasets has proven to be important for supporting diverse use cases like interlink maintenance and synchronization of dataset versions, replicas, and interconnected datasets [1], [5]. However, existing change detection mechanisms have certain limitations in our opinion (see below).

These limitations have motivated our research in dataset dynamics: **(a) Change classification limitation** – The existing change classification denotes the resources that have had their IRI changed as "moved" or "renamed" resources. These resources can lead to structurally broken interlinks[1], which can be repaired by redirecting the old IRI to the new IRI. However, a subset of these "moved" resources may have changes in both their IRI and representation. Structural repair of broken interlinks related to these resources will make them potential candidates for semantically broken interlinks (an interlink is semantically broken when the meaning of the representation of target differs from the intended meaning of the source [3]). This is because iterative updates in a resource of linked dataset (case for many linked dataset [13]) may lead to the semantic drift in the original representation of the resource [14]. Hence, it is important to identify the resources with both changed IRI and representation as a separate class of changes; **(b) Properties selection limitation** – The existing approaches either use *pre-defined properties* or all the *object properties* (graph structure) of a resource to identify the resources that have changed their IRIs and possibly their representation (structure and/or values) too. Selection of pre-defined properties requires additional effort (pre-change detection) which can be saved by using the graph structure. But the disambiguation of resources using the graph structure alone could be challenging [12]. The accuracy of algorithms which rely on the graph structure can be improved by incorporating the data properties of resources in the matching process; **(c) Change model[2] limitation** – The state-of-art has several change models that represent either changed resources or changed triples in a linked dataset. These change models do not allow to identify a changed resource along with its changed triples. The existence of this gap prevents a single change detection mechanism from supporting diverse use cases like interlink maintenance and replica synchronization is addressed in change models like Roussakis et al. [5]. These change models are generally designed by using multiple semantic layers of changes for e.g. simple changes (adds semantic to each added/ deleted triple); complex changes (add further sematic to simple changes by grouping multiple simple changes). Such change models represent changes in more human understandable manner. However to *automatically* support use case like interlink maintenance, these models can further be simplified to incorporate one semantic layer of changes, i.e. the resource-level changes, as demonstrated by Popitsch et al. [1].

The specific *research question* under evaluation in this paper is to what extent we can detect, classify, and model the changes between two versions of a linked dataset?

The contribution of this paper is as follows: *Firstly*, it proposes **DELTA-LD**, a novel change detection and classification approach for linked datasets. DELTA-LD addresses the limitations **(a)** and **(b)** described above. *Secondly*, it proposes the **DELTA-LD change model** to represent detected changes, addressing limitation **(c)**.

[1]An interlink is structurally broken, if either source or target are no longer dereferenceable.
[2]The generic model that represent detected changes and can be materialized in any schema.

Thirdly, it evaluates the accuracy of DELTA-LD using real world data and compares the approach to state-of-art change detection approaches.

The paper is organized as follows: **Section 2** discusses how the aforementioned limitations exist in state of the art approaches. **Section 3** describes the proposed DELTA-LD change model. **Section 4** describes the proposed DELTA-LD approach. **Section 5** presents an evaluation of DELTA-LD in terms of accuracy achieved. **Section 6** discusses a case study that was performed to repair structurally broken interlinks using the change information generated by DELTA-LD and represented in DELTA-LD change model. Conclusions are drawn in **Section 7**.

2 RELATED WORK

This paper aims to address the use case of broken interlinks maintenance and replica synchronization of a linked dataset by identifying the changes between two versions of the dataset. Thus, the change detection approaches that support these use cases and their limitations are discussed in this section. For better clarity, first the types of change information required for each use case are explained, then the change detection approaches that support these use cases are discussed.

UC 1a - Semantic link maintenance – detection of semantically broken interlinks require identifying the updated resources[3]. **UC 1b - Structural link maintenance** – detection of structurally broken interlinks require to identifying the renamed[4] or deleted[5] resources. **UC 1a** and **1b** need the information about the changed resources thus, *resource level change information* is required to support these use cases. **UC 2 - Replica synchronization** – to synchronize a replica of a linked dataset, it is important to identify the added and deleted triples in the dataset. Thus, *triple level change information* is required to support this use case.

Approaches [1], [5], [6], and [15] identify resource level change information. The approach [6] detects the renamed resources to identify and repair structurally broken interlinks. However, this approach partially detects structurally broken interlinks, as the approach does not attempt to identify deleted resources. A SPARQL based change detection approach [5] for linked datasets was proposed to capture human understandable changes. For this, the approach first identifies simple changes by adding semantics to triple level changes, and to add further semantics, the approach groups multiple simple changes into complex changes. The approach can support UC 2 using simple changes. Since, no heuristic based matching is involved in the approach, it would be difficult to identify the resources that have had both their IRIs and representation changed at the same time, thus, the approach supports UC 1a and 1b partially. Another approach [15] targets to capture human understandable changes. The approach detects 132 types of change and can support UC 1a, 1b, and 2. UC 2 can be supported by using basic changes (triple-level changes). For use case 1a, changes like Retype_Individual_To_Individual (change in rdf:type of an instance) can be used to identify updated resources in linked dataset. To support UC 1b, the

[3] Resources that have had their representation changed during the changes in Linked Datasets

[4] Resources that have had their IRI changed during the changes in Linked Datasets.

[5] Resources that have been removed from the dataset during the changes in Linked Datasets.

approach detects renamed resources by calculating the similarity between the neighborhood (graph nodes of the observed resources) of two resources with different IRIs in different versions. However, sometimes the renamed resources may also get updated at the same time. So, with this classification, the resources that have had both their IRIs and representation changed cannot be differentiated from the resources that have had only their IRIs changed (see change classification limitation in the Introduction). The approach [1] supports UC 1a and 1b and classifies the resource level changes as follows: *create* (addition of a new resource in linked dataset); *remove* (same as deleted resource); *update* (described above); *move* (same as rename). Analogous to the classification by approach [15], the distinction between the resources that have had both their IRIs and representation changed, and the resources that have had only their IRIs changed, will be difficult with the classification by approach [1]. This is because former and latter are denoted by just single type of change, i.e. "move" type of change.

Out of all approaches discussed above, only [1], [6], and [15] can identify moved/ renamed resources. However, [1] requires pre-determined properties, while [6] and [15] rely on the graph structure of resources to detect moved resources. This limitation of property selection is referred to in the Introduction as limitation (b).

[7] proposes a triple level change detection approach for RDF datasets. Another approach [9], also identifies triple level changes in RDF datasets by considering the existence of blank nodes in RDF datasets. The approaches in [7] and [9] identify the triple level changes, thus, are suitable to support UC 2.

In this section so far, we identified that the state-of-art approaches detect changes at two levels, triple and resource, with the different levels supporting different use cases. To the best of our knowledge, few existing approaches detect and represent change information at both resource and triple levels. However, we believe that these representations can further be simplified in the context of use cases like 1a, 1b, and 2.

Out of the approaches discussed above, only [1], [5], and [7] proposed models to represent changes in linked datasets. The change model by approach [5] has two semantic layers of change (explained above): simple changes; complex changes. For UC 1a, 1b, and 2, this model can further be simplified and modeled by just one semantic layer of change (resource level changes), which is sufficient to identify a resource level change with its corresponding triples. The single semantic layer change model is already used by [1] for use case 1b, where the changed triples are associated with their corresponding changed resources. However, [1] did not include the changed triples separately as added and deleted triples, which projects the gap between resource level and triple level changes. This is discussed in change model limitation in the Introduction. The ontology proposed by approach [7] bridges the gap between resource and triple-centric view of changes by reification of triples. In reification, a "type" is added to each added and deleted triple, which includes resource level change information. However, this will generate redundant information for the resource level changes as a resource constitutes multiple triples. For UC 1a and 1b, where resource-level changes are required, using such change model would require further operations over generated delta to overcome the redundant resource level change information.

This section has illustrated that the limitations mentioned in the introduction exist in the state-of-art approaches. We argue in this paper that in order to address these limitations we need to: (a) separate by means of classification, resources that only

change their IRIs and resources that both change their IRIs and representation; (b) a change detection mechanism that does not require pre-determined properties for the identification of moved resources; (c) an improved yet simplified change model that bridges the gap between resource and triple centric view of changes.

3 DELTA-LD CHANGE MODEL

We propose the **DELTA-LD Change Model** to address limitation (c) presented in the Introduction. The DELTA-LD change model is based on the Layered Change Log model [10] that models change in ontology at two levels of granularity: *first level* models the information of atomic level change operations; *second level* models the objective of the atomic changes. A similar model can be applied to represent the changes in linked datasets, which will bridge the gap between the resource and triple level changes using just one semantic layer of changes (resource level changes). Analogous to [10], we model the deleted and added triples as atomic operations at the first level, while at the second level the objective of deleting or adding a triple is modeled as the creation, removal, update, movement, or renewal of a resource. Fig. 1 describes DELTA-LD change model. This is a generic model that can be realized/materialised in ontologies, XML schema, or any other type of vocabulary.

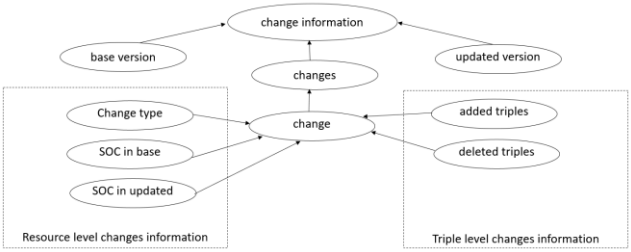

Fig. 1. DELTA-LD change model

In Fig. 1, the "base version" and the "updated version" entities are the older and newer version of a linked dataset used for change detection; the "change type" entity represents the type of resource level changes i.e. *create*, *remove*, *update*, *move*, *renew*; "SOC in base" and "SOC in updated" entities represent subject of change (resource IRI) in base and updated version respectively; finally, "removed triples" and "added triples" entities are used to represent information of deleted and added triples respectively. Using such a change model, one can identify the resource level changes in linked datasets along with their corresponding added and deleted triples.

4 DELTA-LD APPROACH

DELTA-LD is an approach that detects changes between two versions of a linked dataset. The main objective of DELTA-LD is to address the limitations (a) and (b) presented in the Introduction. To address limitation (a), DELTA-LD proposes a sub-

classification for move/ rename type of changes. The proposed sub-classification has following types of changes: *Move:* change only in the IRI of an existing resource; *Renew:* change in the IRI as well as in the representation of an existing resource. Apart from these two types of change, DELTA-LD also detects *create*, *remove*, and *update* type of changes, which are explained in section 2. To address limitation (b), DELTA-LD first identifies all the potential moved and renewed resources by comparing all the features (section 4.2) of removed resources with the features of created resources. Then to filter out the incorrect moved or renewed resources, DELTA-LD determines critical features (section 4.3) automatically. In the context of DELTA-LD, a feature is critical, if its value remains same for most (configurable) of the moved and renewed resources in both versions of a linked dataset.

To implement DELTA-LD initially, we used a triple store and a database. The Triple store is used to upload the dumps of older and newer versions of a linked dataset, and the database is used to store the features (derived from properties and objects) of the resources to facilitate the identification of moved and renewed resources. The approach contains four major activities (see Fig. 2). First is the **Ingestion activity** (section 4.1) – it uploads both the older and newer versions of a linked dataset in a triple store. Second is the **Feature extraction activity** (section 4.2) – this extracts the properties and objects of newly added and deleted resources identified by comparing older and newer version. Third is the **Change detection and classification activity** (section 4.3) – it uses the extracted properties and objects of newly added and deleted resources to identify the updated, moved and renewed resources. Fourth is the **Transformation activity** (section 4.4) – it transforms the identified change information into RDF according to the DELTA-LD change model. The following sub-sections describe each of these activities in more detail.

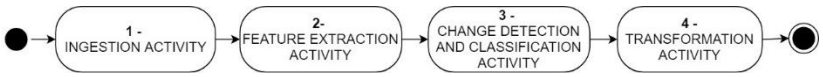

Fig. 2. Overview of the major activities in the DELTA-LD approach

4.1 INGESTION ACTIVITY

Requirements: The aim of this activity is to upload the RDF dataset dumps in a triple store, in a way that the different dumps can be distinctly identified in the triple store.

Implementation: We have implemented an ingestion component which uploads RDF dataset dumps for both older and newer version in a triple store, in two distinct graphs. We have used the *sem:rdf-load()* function of the semantic API of the MarkLogic platform[6] to upload the dumps in the MarkLogic triple store[7].

4.2 FEATURE EXTRACTION ACTIVITY

Requirements: The aim of this activity is to generate features for each newly added and deleted resources in the uploaded datasets using their properties and objects. The

[6] https://www.marklogic.com/

[7] https://www.marklogic.com/product/marklogic-database-overview/

generated features are used for detection of the same resources having different IRIs and similar representation (but not the same) in different versions of a linked dataset.

Implementation: To address the requirement, we implemented a feature extraction component. A resource could correspond to multiple triples. These triples contain various properties and objects that define a resource. To identify the same resources having different IRIs and similar representation (but not the same) in older and newer version of a linked dataset, the properties and objects of newly added resources are compared with the properties and objects of the deleted resources. SPARQL queries described in Listing 1 are used to identify the newly added and deleted resources between the older and newer version of the dataset.

Listing 1. SPARQL queries to identify added (a) and deleted resources(b)

```
                    (a)
SELECT DISTINCT ?S
WHERE { GRAPH <$NEWER-VERSION>
        {?S ?P ?O }
         FILTER NOT EXISTS
         { GRAPH <$OLDER-GRAPH>
         {?S ?P ?O}
      }}
```

```
                    (b)
SELECT DISTINCT ?S
WHERE { GRAPH <$OLDER-VERSION>
        {?S ?P ?O }
         FILTER NOT EXISTS
         { GRAPH <$NEWER-GRAPH>
         {?S ?P ?O}
     }}
```

The updated resources are also identified as both added and deleted resources by checking the IRI of identified added resources (by query (a)) in older version and the IRI of identified deleted resources (by query (b)) in newer version of the dataset.

To identify the same resources having different IRIs in older and newer versions of a linked dataset, we need to consider that it is not necessary that the properties and objects corresponding to the same resource with different IRIs in older and newer versions remain same. Due to the evolution of the dataset, new properties might be added; existing properties might be deleted; object value could be modified. In such scenarios, a similarity between two resources need to be calculated.

To facilitate the similarity calculation, all the properties and objects of the newly added and deleted resources are used to create Feature XMLs (fig 3). While creating Feature XMLs, the component also allows to use additional information[8] about the resources, which is present outside the older and newer versions uploaded during the ingestion activity (Section 4.1).

```
<allFeatures res="http://dbpedia.org/resource/Hamid_Taqvaee" state="delete">
  <givenname type="text">aihhmtd</givenname>
  <name type="text">aihhmtdaettkfe</name>
  <surname type="text">aettkfe</surname>
  <subject type="numeric">1949_births</subject>
  <subject type="uri">sharif_university_of_technology_alumni</subject>
  <subject type="uri">worker-communist_party_of_iran_politicians</subject>
</allFeatures>
```

Fig. 3. Features XML for a resource

A Feature XML represents a resource. The root element is arbitrarily named as “allFeatures” and it contains an attribute (“res”) to link the Feature XML with original resource by keeping the IRI of the resource as the value of “res” attribute. The child element of the root element has two parts: **feature** (name of element), i.e. one of the property name of a resource; **feature value** (value of element), i.e. transformed object value, which is denoted by “key”. The keys are created to cope with the potential errors generally made by humans at data entry stage [8]. The keys are formed

[8] This additional information could be categories, provenance, archival mementos, etc.

differently for different types of object values: *URIs (object properties that contain no numeric character):* The algorithm uses the last token (tokenize with '/') of URI path as the key; *text (data properties that contains no numeric character):* To create the key for each word in the text, the algorithm takes all distinct vowels, first and last character, and the primary key of double metaphone[9] encoding of the word, this combination has been used to improve the accuracy of the algorithm while calculating the similarity of two resources [11]. For instance, the key for "Hamid" will be "aihhmtd", where "ai" are distinct vowels, "h" is the first character, "hmt" is the primary key of the double metaphone encoding, and "d" is the last character of the word. *Numbers/ string with digits:* the algorithm uses the exact value of the objects as key. Both parts of a feature are utilized at later stage by DELTA-LD to identify critical features (section 4.3). The component stores the Features XMLs of newly added and deleted resources in "new" and "delete" collections[10] of MarkLogic database[11] respectively. We used a database for storing Feature XML as in MarkLogic, a separate installation is not needed for database because both triple store and database in MarkLogic are physically same but conceptually different.

XQuery (language to query XML data) is the language to access the data stored in MarkLogic database, thus, we chose to keep the information of features in XML format. One can choose a different format and storage mechanism to store features. For e.g. features can also be stored in a triple format in a triple store.

4.3 CHANGE DETECTION AND CLASSIFICATION ACTIVITY

Requirement: We need an activity to: **(a)** classify *updated* resources; **(b)** identify resources that have had only their IRIs changed and classify them as *move* **(c)** identify resources that have had both their IRIs and representation changed at the same time and classify them as *renew*.

Implementation: The state-of-art approaches compare specific properties and corresponding objects of a resource in the older and newer versions to detect the resources that have different IRIs in these versions (*move* and *renew* types of change). The selection of these properties is determined by their coverage[12], which implies that the property name alone is used as a criterion for its selection. This is irrespective of the object values of the properties which may have changed by the evolution of the dataset. However, the DELTA-LD approach differs from these techniques in detecting the *move* and *renew* type of changes as it relies on critical features to produce more accurate results. The selection process of critical features uses both the feature name (property name) and feature value/ key (derived from object value).

To fulfill requirement (a), the component identifies resource IRIs that have Features XML in both new and delete collection. Triples corresponding to these IRIs are extracted and compared to identify the added and deleted triples. The information of these resource IRIs, along with their added and deleted triples is stored in "update"

[9] http://www.b-eye-network.com/view/1596
[10] https://docs.marklogic.com/guide/search-dev/collections
[11] https://www.marklogic.com/
[12] Coverage of a property is the number of resources with triples containing that property.

collection. Finally, the older XML (Feature XML in delete collection) and the newer XML (Feature XML in new collection) are deleted.

For requirement (b), the component identifies the moved resources by matching the remaining older XMLs with the remaining newer XMLs. A pair of older and newer XML is decided as a *match*, when both XMLs share similar features. The component incorporates three configuration parameters to facilitate matching: *accept threshold:* a match having confidence[13] equal or above this threshold is selected as an authentic match; *audit threshold:* a match having confidence above this threshold and lower than the accept threshold goes to the audit routine, which decides the authenticity of the match; *critical feature threshold:* If the rate of a feature and its value being same in the matches (confidence > accept threshold) is greater than the critical feature threshold, then the feature is identified as critical. The component identifies the critical features by comparing each feature (with value) of the older and newer XMLs of the matches having confidence greater than the accept threshold. Subsequently, to decide the authenticity of the matches having confidence between accept and audit threshold, the *audit routine* ensures that the critical features and their values must be same in the matched older and newer XMLs. For e.g., assume "name" is identified as a critical feature, then audit routine checks whether a match (accept threshold > confidence > audit threshold) has same values for feature "name" in both older and newer XML. If same value is not found in both XMLs then audit routine discards the match, else it is selected as an authentic match. The component then stores the resource IRIs of the matched older and newer XMLs as moved resources in "move" collection. Finally, the matched XMLs get deleted from the delete and new collection. The resources related to the remaining Features XMLs in delete and new collection are classified as the *deleted* and *created* resources respectively. Fig. 4 provides a brief description of all the steps performed by this component to address requirement (b).

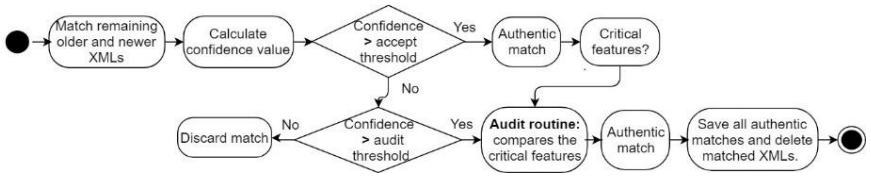

Fig. 4. Requirement (b) - identification and classification of moved resources

For requirement (c), the component extracts properties and objects of the detected moved resources from older and newer version of the dataset, and compares them. The component shifts the information of all the moved resources from "move" to "renew" collection, for which there is any added/ deleted property or updated object.

4.4 TRANSFORMATION ACTIVTY

Requirement: The aim of this activity is to represent the information of the classified changes in RDF according to the DELTA-LD change model.

[13] The confidence is calculated by the following formula: ((no of features matched between the older and newer XML) / (no. of features in the older XML)) * 100

Implementation: The component sequentially accesses and transforms the information in each new, delete, update, move, and renew collection to RDF according to the ontology (available online[14]), which has been created based on the DELTA-LD change model. Finally, the transformed information is stored in distinct graphs in the triple store according to the collection name.

5 EVALUATION – Accuracy of DELTA-LD implementation

The purpose of this experiment was to determine the accuracy of initial DELTA-LD implementation and compare it with existing approaches [1] and [6]. To the best of our knowledge, [1] and [6] use similar change metrics as DELTA-LD and used the same experimental datasets to perform their experiment, leading to their selection for comparison. *The hypothesis of the experiment was that the accuracy of DELTA-LD would be better than [1] and [6] in terms of F-measure.*

Datasets[15]: The experiment has been conducted on two different sets of input. Each set of input contains two versions of a linked dataset, additional information datasets, and a gold standard to determine the accuracy of DELTA-LD.

First set: For change detection, we used the enriched DBpedia person snapshots 3.2 and 3.3 (**20,284** and **29,498** resources) provided by [1]. For the resources in snapshots 3.2 and 3.3, we have used the additional information present in the article category dataset 3.2 and 3.3 respectively. The article category dataset of DBpedia groups the resources of same category, using SKOS and Dublin core vocabularies. These datasets are chosen for additional information due to following reasons: i) to demonstrate that DELTA-LD is easily extensible to include additional information; ii) [6] also included the article category dataset to identify moved resources, which gives us an opportunity to discuss the impact of critical features on the results. The gold standard used to determine the accuracy of DELTA-LD was provided by [1] and contains 179 move type of changes. On analyzing our results, we found 1 *moved* resource[16], that is not covered by the gold standard. Hence, we increased the *move* type of changes in the gold standard to **180**. Also, **5666** resources were excluded by [1] for detecting changes and are mentioned as "unknown-created" and "unknown-removed" in the gold standard. For same baseline, we too omitted these resources.

Second set: We executed DELTA-LD with DBpedia person snapshot 3.6 and 3.7 (**296,595** and **790,703** resources). These datasets are much bigger than the datasets in the first set. Again, for the additional information, the corresponding article category datasets were used. To the best of our knowledge, a gold standard for the changes between these versions is not available as yet. Thus, to determine the accuracy of DELTA-LD, we created a gold standard for the resources that changed their IRIs or both IRIs and representation, using the following three steps: *Step 1*, We used DBpedia redirect dataset version 3.7 and extracted the redirects in which the source

[14] https://github.com/anujsinghdm/DELTA-LD/blob/master/schema.owl

[15] Links for the respective datasets mentioned in this section are stated in a file available online (https://github.com/anujsinghdm/DELTA-LD/blob/master/DSLinks.txt)

[16] Older: http://dbpedia.org/resource/Kim_Jin-Kyu; newer: http://dbpedia.org/resource/Kim_Jin-Kyu_%28football_player%29

IRI is present in person snapshot 3.6 and target IRI is present in the person snapshot 3.7. We identified **3390** redirects this way. These redirects can be treated as the resources that changed their IRIs. *Step 2,* During the analysis of our results, we found some *move* and *renew* type of changes in the DBpedia disambiguation dataset version 3.7 where some of the IRIs of person snapshot 3.6 are linked with one or more different IRIs of the person snapshot 3.7. However, there is only one link between older resource IRI and newer resource IRI which denotes that both older and newer resource are same, but just that the older resource has been moved to a different IRI. We filtered out **585** links this way. *Step 3,* We manually verified *move* and *renew* types of changes in our results that are not present in the gold standard prepared in Step 1 and 2; and found **296** instances correctly detected by DELTA-LD.

Experimental method: We conducted the experiment on a machine having 7th generation i7 processor with 16 GB RAM in two stages. *First stage* – DELTA-LD was executed with the datasets of the first input set. For this, we uploaded the person snapshot 3.2 and 3.3 and their corresponding additional information in MarkLogic triple store in four distinct graphs. We created the features of the resources in the person snapshots using the feature extraction component (Section 3.2). Subsequently, change detection mechanism was invoked using the following three configuration parameters: accept threshold – 80%; audit threshold – 40%; critical feature threshold – 98%. Accept and audit thresholds were selected according to [1]. For critical feature threshold, it was intuitive to keep the maximum value, but a room of 2% was given for exceptions. This resulted in classified changes between person snapshots. To determine the accuracy of DELTA-LD, the results were first compared with the gold standard, and extra changes detected by DELTA-LD were manually verified. *Second stage* – the same steps from the first stage were executed on the second set of input.

Table 1: Detected changes for first and second set and their stats

Type of stats	Create	Remove	Update	Move	Renew
Number of changes for first input set	**3819**	**239**	**4161**	**124**	**46**
% of total changes in first set	45.5%	2.8%	49.6%	1.47%	0.54%
% of changed resources in first set	12.9%	1.17%	20.5%	0.61%	0.22%
Number of changes for second input set	**499590**	**5482**	**50380**	**2723**	**1529**
% of total changes in second set	89.2%	0.97%	9.0%	0.48%	0.27%
% of changed resources in second set	63.1%	1.84%	16.98%	0.91%	0.51%

Results: Table 1 describes the classification of changes detected by DELTA-LD for both first and second input set. For first set, we identified 3819 resources that were newly added in snapshot 3.3 and 239 resources in snapshot 3.2 that were not included in snapshot 3.3. We also identified that the representation of 4161 resources has been changed from snapshot 3.2 to 3.3. Additionally, 124 resources had different IRIs in snapshot 3.2 than in snapshot 3.3. Furthermore, we detected 46 *renewed* resources that are special type of resources for which both the IRI and representation have changed from snapshot 3.2 to 3.3. These resources were detected by the sub-classification of *move*/ *rename* type of changes in the existing classifications. Similarly, we detected and classified changes for second set, the number of which is much greater than the changes detected for first set because the second set is much wider than first set. For second set, we identified 1529 *renewed* resources.

As shown in table 1, for first set of input, the majority of changes belong to *create* and *update* type of changes, i.e. 45.5% and 49.6% respectively. However, the share of

create increases in the second set by 43.75%, which can also be confirmed by a big difference in the resources present in person snapshots 3.6 and 3.7. On the other hand, the share of update decreases in the second set by 40.6%. *Remove* type of change captures a lower percentage of the total changes for first set, i.e. 2.84%, which further decreases by 1.87% for second set. However, the total resources (1.84%) that were removed from person snapshot 3.6 (older version in second input set) is still greater than the total removed resources (1.17%) from person snapshot 3.2 (older version in first input set). The percentage shares for *move* and *renew* type of changes for first input set are 1.47% and 0.54% respectively. The percentage shares for *move* and *renew* changes decreases by 1% and 0.27% respectively for second input set. The results suggest that rate of adding new resources in DBpedia dataset increases from version 3.3 to version 3.7. The rate of updating resources in DBpedia reduces from version 3.3 to 3.7, as 20.5% resources of person snapshot 3.2 were updated in person snapshot 3.3, while only 16.9% resources in person snapshot 3.6 were updated in snapshot 3.7. Though the moved and renewed resources are low in percentage, the results show that this number tend to increase with evolution of DBpedia dataset.

The state-of-art emphasizes on determining the accuracy of a change detection approach using the resources that have had their IRIs changed. These constitute the *move* and *renew* type of changes detected by DELTA-LD. Since, the gold standard does not have these changes separately, but as *move* type of changes, we combined the DELTA-LD results for *move* and *renew* types of changes as *move* type of changes. Table 2 describes the accuracy of DELTA-LD in detecting *move* type of changes. For the first set of input, the precision of DELTA-LD is 1, as DELTA-LD ensures that the critical features are same in the low confidence matches. The approach was not able to detect 10 *moved* resources. Out of the 10, 5 *moved* resources were rejected by the audit routine as the matched older and newer Features XMLs did not have the same critical features. The other 5 had one-to-many matches, i.e. one older Features XML was matched with more than one newer Features XML and hence discarded. The calculated F-measure for the first input set is 0.9714.

The precision reduced for second input set due to two types of incorrectly detected *move* type of changes, *move* with higher confidence; *move* with lower confidence (< 50). Majority (~50%) of incorrect *move* belong to the latter. *Move* with such low confidence were not found for first input set. This implies, to increase the audit threshold from 40% to 50% by which the precision can be increased by ~2% with slight decrease in recall by ~0.1%. For incorrect *move* with higher confidence, the matched older and newer Features XMLs share a high percentage of same features, including critical features. The recall for second input set is greater than the recall for first due to the presence of wider information about the resources in second input set, which led to more one-to-one matches between older and newer Features XML.

Table 2. Accuracy of DELTA-LD in detecting moved resources

Input set	Move (gold standard)	Move (DELTA-LD)	Precision	Recall	F-measure
First	180	170	1	0.9444	0.9714
Second	4271	4252	0.9597	0.9555	0.9576

The approaches [1] and [6] published their detected changes for the snapshots of first input set, which we used to compare the accuracy of DELTA-LD with these approaches (Table 3). The approach [1] used various configuration properties to

identify the moved resources; with a maximum recorded precision (1.0), recall (~0.91), and F-measure (~0.95) using the foaf:name property. The approach [6] identified moved resources with a precision ~0.87, recall ~0.99 and F-measure ~0.93. A probable reason for low precision but good recall can be the use of object properties (graph structure of resources) only to identify moved resources. DELTA-LD uses the same object properties through additional information to create features, but achieves much better precision by selecting critical features out of all the features.

Table 3: Accuracy of Popitsch et al., Pourzaferani et al., and DELTA-LD

Accuracy	Popitsch et al.	Pourzaferani et al.	DELTA-LD
Precision	1	0.87	1
Recall	0.91	0.99	0.9444
F-Measure	0.95	0.93	**0.9714**

By comparing our results of first input set with the results of approaches [1] and [6], it has been identified that DELTA-LD outperforms [1] by ~2% and [6] by ~4% in terms of F-measure. Hence, it can be argued that the results support our hypothesis - that the accuracy of DELTA-LD would be better than [1] and [6] in terms of F-measure. We were not able to compare the results of the second input set with any other approach, as to the best of our knowledge, only [6] has published the results of their detected moved resources between DBpedia person snapshot 3.6 and 3.7. However, the gold standard used by [6] has a different count than the gold standard used by us. [6] neither contains information about how precisely they created their gold standard nor have they published their gold standard as yet. This prevents us from comparing the results of second input set at this time.

To support the evaluation of the change detection approaches in future, the gold standard prepared by us for the moved and renewed resources between DBpedia person snapshot 3.6 and 3.7 is available online[17].

6 CASE STUDY: Repair of broken interlinks of DBpedia

The changes detected by DELTA-LD and modeled by DELTA-LD change model can be used to automatically repair structurally broken interlinks. To demonstrate this, we repaired and validated the structurally broken interlinks in source DBpedia person snapshot 3.7 to target Freebase. The steps for the case study are outlined in Fig. 5.

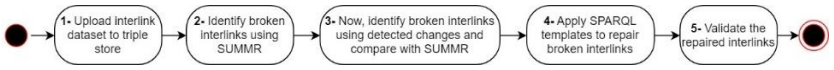

Fig. 5. Sequence of steps to repair and validate the broken interlinks

Step 1, the interlink dataset[18] with links from source to target is uploaded to MarkLogic triple store in a distinct graph. *Step 2*, before repair the interlinks, we identified 704 structurally broken interlinks between source and target using a separate approach, namely SUMMR validation template [4]. *Step 3*, we consumed the changes detected by DELTA-LD for second input set to identify the broken interlinks

[17] https://github.com/anujsinghdm/DELTA-LD/blob/master/GS.xml
[18] http://oldwiki.dbpedia.org/Downloads37#linkstofreebase

using SPARQL templates (available online[19]) developed for the case study. The same number of broken interlinks as step 2 were identified. However, in step 3, we were also able to identify the reason for broken interlinks, *specifically 656, 17, 31 interlinks were broken due to the removed, moved, and renewed resources* respectively. *Step 4,* the identified broken interlinks were repaired using SPARQL templates (available online[20]) developed in the case study. To repair, the templates only delete the broken interlinks of removed resources. For broken interlinks of moved and renewed resources, the template first deletes the broken interlinks, then adds a new interlink using the IRI of the newer resource. The template deleted 656 broken interlinks of the removed resources. Out of 17 broken interlinks of moved resources, the template identified that 12 repaired interlinks (interlinks using the IRI of the newer resources) were already present in the interlink dataset. So, the template deleted all 17 broken interlinks, but added only 5 new interlinks in the interlink dataset. For 31 broken interlinks of renewed resources, all the repaired interlinks were already present in the interlink dataset. Hence, 31 broken interlinks were deleted but no new interlink was added. *Step 5*, after repair we have again used the SUMMR template to identify broken interlinks, which identified 0 broken interlink this time.

In future, we intend to demonstrate the use of DELTA-LD (approach and change model) for other use cases mentioned in the related work.

7 CONCLUSION

The paper presents DELTA-LD approach to detect and classify the changes between two versions of a linked dataset along with DELTA-LD change model for modeling the detected changes. The *research question* in this paper was to what extent we can detect, classify, and model the changes between two versions of a linked dataset?

To answer the research question, we executed DELTA-LD on DBpedia person snapshots 3.2 and 3.3 (20,284 and 29,498 resources), and DBpedia person snapshots 3.6 and 3.7 (296,595 and 790,703 resources). The approach detected created, removed, updated, moved, and renewed resources, which are (3819, 239, 4161, 124, 46) and (499590, 5482, 50380, 2723, 1529) for former and latter set of snapshots respectively. The results support the hypothesis, as DELTA-LD outperforms the state-of-art approaches [1] and [6] by ~2 - 4 % in terms of F-measure. Representing *renew* as a separate class of change identified up to 1529 additional resources that changed their representation. We also presented DELTA-LD change model that allows to view a changed resource along with its added/ deleted triples at the same time. Finally, we presented a case study to repair structurally broken interlinks from DBpedia person snapshot 3.7 to Freebase, using the changes detected between DBpedia person snapshot 3.6 and 3.7; 704 structurally broken interlinks were repaired and validated.

Acknowledgments. This research has received funding from the ADAPT Centre for Digital Content Technology, funded under the SFI Research Centres Programme (Grant 13/RC/2106) and co-funded by the European Regional Development Fund.

[19] https://github.com/anujsinghdm/DELTA-LD/tree/master/IBI
[20] https://github.com/anujsinghdm/DELTA-LD/tree/master/RBI

References

1. Popitsch, N. and Haslhofer, B., 2011. 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), pp.266-283.
2. Umbrich, J., Villazón-Terrazas, B. and Hausenblas, M., 2010. Dataset dynamics compendium: A comparative study.
3. Kovilakath, V.P. and Kumar, S.D., 2012, August. 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.
4. Meehan, A., Kontokostas, D., Freudenberg, M., Brennan, R. and O'Sullivan, D., 2016, October. Validating Interlinks Between Linked Data Datasets with the SUMMR Methodology. In OTM Confederated International Conferences" On the Move to Meaningful Internet Systems" (pp. 654-672). Springer, Cham.
5. Roussakis, Y., Chrysakis, I., Stefanidis, K., Flouris, G. and Stavrakas, Y., 2015, October. A flexible framework for understanding the dynamics of evolving RDF datasets. In International Semantic Web Conference (pp. 495-512). Springer, Cham.
6. Pourzaferani, M. and Nematbakhsh, M.A., 2013. Repairing broken RDF links in the web of data. International Journal of Web Engineering and Technology, 8(4), pp.395-411
7. Pernelle, N., Saïs, F., Mercier, D. and Thuraisamy, S., 2016. RDF data evolution: efficient detection and semantic representation of changes. Semantic Systems-SEMANTiCS2016, pp.4-pages.
8. Ward, D.J., Blackwell, A.F. and MacKay, D.J., 2000, November. Dasher—a data entry interface using continuous gestures and language models. In Proceedings of the 13th annual ACM symposium on User interface software and technology (pp. 129-137). ACM.
9. Lee, T., Im, D.H. and Won, J., 2016. Similarity-based Change Detection for RDF in MapReduce. Procedia Computer Science, 91, pp.789-797.
10. Javed, M., Abgaz, Y.M. and Pahl, C., 2014. Layered change log model: bridging between ontology change representation and pattern mining. International Journal of Metadata, Semantics and Ontologies, 9(3), pp.184-192.
11. Patrick, J., Sabbagh, M., Jain, S. and Zheng, H., 2010. Spelling correction in clinical notes with emphasis on first suggestion accuracy. In Proceedings of 2nd Workshop on Building and Evaluating Resources for Biomedical Text Mining (BioTxtM2010) (pp. 1-8).
12. Zheng, Z., Si, X., Li, F., Chang, E.Y. and Zhu, X., 2012, December. Entity disambiguation with freebase. In Proceedings of the The 2012 IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technology-Volume 01 (pp. 82-89). IEEE Computer Society.
13. Gerber, D. and Ngomo, A.C.N., 2011. Bootstrapping the linked data web. In 1st Workshop on Web Scale Knowledge Extraction@ ISWC (Vol. 2011).
14. Li, Z., He, Y., Gu, B., Liu, A. and Zhou, X., 2017. Diagnosing and Minimizing Semantic Drift in Iterative Bootstrapping Extraction. IEEE Transactions on Knowledge and Data Engineering.
15. Papavasileiou, V., Flouris, G., Fundulaki, I., Kotzinos, D. and Christophides, V., 2013. High-level change detection in RDF (S) KBs. ACM Transactions on Database Systems (TODS), 38(1), p.1.