

Data-driven Assessment of Structural Evolution of RDF Graphs

Carlos Bobed [a,b,*], Pierre Maillot [c] and Peggy Cellier [c] and Sébastien Ferré [c]
[a] *everis / NTT Data*
[b] *University of Zaragoza,Spain*
E-mail: cbobed@unizar.es
[c] *Université de Rennes, CNRS, IRISA, France*
E-mails: pierre.maillot@irisa.fr, peggy.cellier@irisa.fr, sebastien.ferre@irisa.fr

Abstract. Since the birth of the Semantic Web, numerous semantic data repositories have appeared. The applications that exploit them rely on the quality of their data through time. In this regard, one of the main dimensions of data quality is conformance to the expected usage of the vocabulary. However, the vocabulary usage (i.e., how classes and properties are actually populated) can vary from one base to another. Moreover, through time, such usage can evolve within a base and diverge from the previous practices. Methods have been proposed to follow the evolution of a knowledge base by following the evolution of their intentional schema (or ontology); however, they do not capture the evolution of their actual data, which can vary greatly in practice.

In this paper, we propose a data-driven approach to assess the global evolution of vocabulary usage in large RDF graphs. Our proposal relies on two structural measures defined at different granularities (dataset vs update), which are based on pattern mining techniques. We have performed a thorough experimentation which shows that our approach is scalable, and can capture structural evolution through time of both synthetic (LUBM) and real knowledge bases (different snapshots and updates of DBpedia).

Keywords: Data Evolution, Data Management, Pattern Mining, Similarity Measure, Semantic Web

1. Introduction

The last years have witnessed a huge growth in the amount of open semantic data, published in Knowledge Bases (KB) such as RDF graphs. Data consumers and application developers are however heavily dependent on the quality of this open data for their usage. One important quality criterion [1] is the stability across time of the usage of the vocabulary (i.e., RDF classes and properties). For example, a movie application will expect a set of properties in the description of films, and will have some of its functionalities become unavailable, or at least degraded, if some properties are removed or replaced during the evolution of the KB.

To manage the evolution of the vocabulary usages in knowledge bases (i.e., how the ontologies forming the vocabulary are actually used and populated), one could suggest that it is enough to compare two sets of properties – one for each version – to assess their evolution. In this regard, several approaches have been proposed to monitor the evolution of ontologies [2, 3], as well as the evolution of their associated data in conformance with their schema [4, 5]. However, a knowledge base may not, in practice, conforms exactly to a fixed schema due to deviations coming either from the content itself, which can have missing data or inadequate vocabulary; or from the heterogeneity of the sources (data publishers and extractors). Indeed, particularly among human sources, the level of modeling skills or the focus of their contributions can also vary. Vocabulary usage should then be seen as a statistical distribution over the sets of classes and properties used in resource descriptions, i.e., over RDF structures. For example, in a movie knowledge base, a possible distribution could be such that 80% of movies have a release date, and 95% of movies with a release date also have a release country. This heterogeneity in vocabu-

*Corresponding author. E-mail: cbobed@unizar.es.

lary usage makes it difficult to detect and measure the structural evolution of RDF graphs, although it must be assessed as it can strongly impact data usage.

Vocabulary usage can be analyzed in terms of *structural patterns*, i.e., combinations of RDF classes and properties, and in terms of the *statistical distribution* of those patterns in the data. Such structural patterns have been shown to be common in RDF graphs [6], and have been used to optimize SPARQL queries [7]. In our previous work [8], we have exploited pattern mining techniques [9] to compare two knowledge bases, measuring their structural similarity. However, this approach only aimed at comparing different knowledge bases, not at evaluating the fine-grained evolution of the structural patterns of a single knowledge base. To that purpose, it is necessary to be able to compare RDF graphs at different levels of detail, i.e., individual updates w.r.t. the entire knowledge base.

In this paper, we propose a data-driven approach to assess the structural evolution of large knowledge bases. When analyzing the evolution of large knowledge bases, such as DBpedia, we can charaterize such an evolution as a sequence of updates (e.g., more than 900,000 updates for DBpedia between 2015-10 and 2016-10). Besides, from time to time, the knowledge base maintainers can publish a version of the KB, called *snapshot*, that represents a consolidated view of the KB. Thus, updates and snapshots are defined at different granularity levels: 1) updates are much more frequent than snapshots; and 2) each update only impacts a small subset of the KB resources while a snapshot contain all resources existing at a point in time.

While we can use our structural similarity measure [8] to compare two snapshots, we need new measures to assess the structural evolution in a fine-grained way, comparing updates to snapshots. In particular, we propose two new measures which makes it possible to: 1) identify which updates are outdated in the sense that they conform to an older snapshot (e.g., they are using old versions of particular URIs, or they follow old modeling practices); and 2) identify which updates alter the heterogeneity of the structural patterns w.r.t. the last snapshot. In this last regard, an increase of heterogeneity means that the update introduces new structural patterns instead of reusing former ones (e.g., adding new properties which were not previously present in the data[1], or deleting properties which are very usual for a given type of resource). while a decrease means that the update introduces structural patterns that are similar to the other structures in the graph (e.g., completing resource descriptions with missing properties). We propose as well to combine the three measures in an assessment framework to help knowledge base maintainers to evaluate the evolution of their knowledge bases. Our approach can account for the situation where the ontology is not exhaustive and groups of resources with similarities appear, for example, in the case of movies from the same country or of the same genre. Based on statistical analysis exploiting the Minimum Description Length (MDL) principle, our proposal can discover those implicit categories (corresponding to the detected structural patterns). Finally, apart from evaluating the measures with synthetic datasets, we have performed a thorough experimentation in a real setting, using snapshots and updates of DBpedia [10]. Observed results indicate that our approach is scalable, and can capture structural evolution through time in a large multi-domain base.

The contributions of the paper are threefold:

- two new similarity measures to compare individual updates to consolidated versions of a KB (snapshots);
- a methodology to apply the different similarity measures to the data-driven assessment of data evolution in a KB;
- an experimental evaluation on both synthetic (LUBM [11]) and real scenarios (DBpedia [10]) of the measures showing the feasibility and scalability of our proposal.

The rest of the paper is as follows. Section 2 briefly describes the basics of the frequent pattern mining techniques that this paper is based on. Section 3 recalls the representations and definitions we need to apply our proposal (i.e., the representation we defined in [8] to use pattern mining techniques to extract structural patterns from RDF graphs, and presents its extension to represent *updates*), as well as a running example which will serve to illustrate the new measures. Then, Section 4 describes the different measures we propose to use to assess the evolution of a knowledge base, and Section 5 introduces the proposed methodology to apply such measures in a deployment scenario. Section 6 details the experiments we have carried out on both synthetic data and DBpedia to show the feasibility of our proposal. Finally, Section 7 discusses the related work, and Section 8 presents the conclusions and future work.

[1] Note that they might even come from a typo in the URIs, as they would be considered as new vocabulary.

2. Preliminaries on Pattern Mining

This section briefly defines the basic concepts of the well-known data mining technique called *frequent itemset mining*. Then, the pattern mining approaches based on the Minimum Description Length (MDL) principle are presented.

2.1. Frequent Itemset Mining

Pattern mining methods are data mining approaches that extract regularities from a database [12]. Formally, let $\mathcal{I}$ be a set of *items*, a *transaction t* is defined as a set of items, and a database $\mathcal{D}$ as a bag of transactions. Table 1 shows an example of a database $\mathcal{D}$ with 6 transactions (t_i) and 5 items (a, b, c, d, e).

Definition 1. *A transaction t* supports *an itemset X iff $X \subseteq t$. The* support *of X in $\mathcal{D}$, $supp_{\mathcal{D}}(X)$, is then the number of transactions of $\mathcal{D}$ that contain X.*

For instance, in Table 1, $supp_{\mathcal{D}}(\{a,b\}) = 3$, which means that {a,b} is supported by t1, t2 and t3.

Definition 2. *An itemset X is called frequent, iff $supp_{\mathcal{D}}(X) \geqslant min_{sup}$ where min_{sup} is a given threshold.*

For instance, in Table 1, for $min_{sup} = 4$, the frequent itemsets are {a} and {b}.

Frequent itemset mining is the pattern mining method that finds all frequent itemsets in a database with respect to a threshold min_{sup}. There exist a lot of algorithms to extract the frequent itemsets from a transactional database (for instance [12–14]).

2.2. Pattern Mining and MDL

One of the major problems with frequent itemset mining, and pattern mining in general, is that the number of extracted patterns can be huge. Several methods tackle this problem, among which one is based on information theory and more specifically on the Minimal Description Length (MDL) principle [9, 15]. The main idea of MDL is that *"the best model compresses the data best"*. From that concept, Vreeken *et al.* [9], propose an approach, called *KRIMP*, to find *"the set of frequent itemsets that yield the best lossless compression of the database"*. Before presenting KRIMP in detail, some notions have to be defined.

Code Table and Standard Code Table A *code table* is a subset of itemsets that can be extracted from a database. An example of a code table CT for the database $\mathcal{D}$ is given in Table 1. In this example, the first column shows the 5 itemsets that have been extracted.

The *standard code table* is a specific code table which contains all and only singletons. An example of a standard code table SCT can be seen as well in Table 1.

Coverage and Usage To each itemset, a *usage* score is associated (see the third column of the code table, Table 1). This score is the number of transactions of the database that are *covered* by this itemset. Indeed *to cover* a transaction, for instance t1 in $\mathcal{D}$, the itemsets of the code table are considered one by one, from the longest to the shortest one, until all items of the transaction are covered by an itemset of the code table without overlapping.

For instance, in the example of Table 1, t1 (from $\mathcal{D}$) is covered by 2 itemsets of the code table CT: {a,b,c} and {d,e} which are denoted by $cover(CT, t1) = \{\{a,b,c\},\{d,e\}\}$. Note that the usage of an itemset can be different to its support, e.g., $supp_{\mathcal{D}}(\{a,b\}) = 3$ whereas $usage_{\mathcal{D}}(\{a,b\}) = 1$. Moreover, note how 2 transactions of $\mathcal{D}$ that contain {a, b} are already covered by {a, b, c}.

Code Length Each selected itemset has also an associated code length which is inversely related to its usage for encoding the database ($usage_{\mathcal{D}}$). The length of the code of an itemset X is defined as:

$$L(code_{CT}(X)) = -log\left(\frac{usage_{\mathcal{D}}(X)}{\sum_{Y \in CT} usage_{D}(Y)}\right)$$

The database is encoded using the code table, by replacing each transaction by the codes of the itemsets that cover it without overlap (i.e., $cover(CT, t)$). The size of the encoded database $\mathcal{D}$ is thus defined as:

$$L(\mathcal{D}|CT) = \sum_{t \in \mathcal{D}} \sum_{X \in cover(CT,t)} L(code_{CT}(X))$$

And the size of the code table CT is defined as:

$$L(CT|\mathcal{D}) = \sum_{X \in CT} L(code_{SCT}(X)) + L(code_{CT}(X))$$

Finally, the total compressed size of the encoded database and the code table is given by:

$$L(\mathcal{D}, CT) = L(\mathcal{D}|CT) + L(CT|\mathcal{D})$$

Table 1
Database $\mathcal{D}$, its code table, its encoded database, and its standard code table.

Database $\mathcal{D}$

transaction	description
t1	{a, b, c, d, e}
t2	{a, b, c}
t3	{a, b}
t4	{a}
t5	{b, d, e}
t6	{d, e}

Code Table CT

itemsets	code	$usage_{\mathcal{D}}$
{a, b, c}		2
{d, e}		3
{a, b}		1
{a}		1
{b}		1

Encoded Database $\mathcal{D}_e$

transactions	encoding
t1	
t2	
t3	
t4	
t5	
t6	

Standard Code Table SCT

itemsets	code	$usage_{\mathcal{D}}$
{a}		4
{b}		4
{c}		2
{d}		3
{e}		3

Compression Ratio The ratio between the length of the encoded database according to the code table and the length of the encoded database according to the standard code table is the compression ratio, denoted by $L\%$:

$$L\% = \frac{L(\mathcal{D}|CT)}{L(\mathcal{D}|SCT)}\ x\ 100$$

The closer this ratio is to 0, the more regular the database is.

KRIMP-based Algorithms The main idea behind KRIMP is to use a greedy algorithm to try to find the code table that compress the database the most. Roughly speaking, the steps of KRIMP over a database $\mathcal{D}$ are as follows. First, its frequent itemsets $\mathcal{F}$ are extracted and are considered as candidates to be part of the final code table. Second, following different heuristics, a subset of $\mathcal{F}$ is selected to form a code table CT such that $L(\mathcal{D}, CT)$ is minimised. While we used the KRIMP algorithm in [8] to compute the code tables, we chose to use a more scalable variant of it, called SLIM [15].

3. Handling the Data

In this section, we first briefly recall the representation of RDF graphs by bags of transactions, proposed in [8], in order to apply frequent pattern mining approaches. Second, we define a transaction-based representation for individual RDF updates. Finally, we present a running example which will serve to illustrate our newly proposed measures.

3.1. From RDF Graphs to Transactions

As seen in Section 2, in order to apply frequent itemset mining approaches on RDF data, we first need to represent them by bags of transactions. Such a representation must be an abstraction of the structural features of the RDF graph that can be useful to detect structural patterns. Although they might incur in some loss of information, this kind of transformations (i.e., *propositionalizations* [16]) are usually successfully applied to enable data mining and machine learning approaches over graphs, as in [17], for example.

In [8], we proposed two representations, namely the *property-based* (PB) representation and the *property-class-based* (PCB) representation, which provide different levels of detail. Both proved to be useful to detect differences in the structure of the graphs. However, we advocate the use of PCB as it provides a finer-grained representation of RDF graphs, and throughout this paper, we adopt it.

The PCB representation is defined at the level of resources, in terms of their types, their outgoing and ingoing triples, as well as the type of their related resources. Each resource is represented by a transaction. The set of items used in transactions is defined as follows.

Definition 3. *Let $\mathcal{B}$ be an RDF graph. Let C and P be respectively the sets of classes and properties used in $\mathcal{B}$. Then $\mathcal{I}_{PCB}$, the set of* items *used in PCB representations, is defined as:*

$$\mathcal{I}_{PCB} = C\ \cup\ (P \times \{in, out\})\ \cup\ (P \times \{in, out\} \times C)$$

In other words, a PCB item represents either: a class c (e.g., "person"), an ingoing property (p, in) (e.g., "is director of"), an outgoing property (p, out) (e.g., "has a birth place"), a qualified ingoing property (p, in, c) (e.g., "is director of a TV Show"), or a qualified outgoing property (p, out, c) (e.g., "has a city as birth place").

The representation of each resource of an RDF graph by a transaction is defined as follows.

Definition 4. *Let r be a resource occurring in an RDF graph $\mathcal{B}$. The PCB representation of r is the transac-*

tion $t_{r,PCB} \subseteq \mathcal{P}(\mathcal{I}_{PCB})$, generated by the application of the following PCB production rules until no new item is added to $t_{r,PCB}$:

(a) $(r,\ \texttt{rdf:type},\ c) \in \mathcal{B} \Rightarrow c \in t_{r,PCB}$
(b) $(r,\ p,\ r') \in \mathcal{B} \Rightarrow (p, out) \in t_{r,PCB}$
(c) $(r',\ p,\ r) \in \mathcal{B} \Rightarrow (p, in) \in t_{r,PCB}$
(d) $\{(r,\ p,\ r'), (r',\ \texttt{rdf:type},\ c)\} \subseteq \mathcal{B} \Rightarrow (p, out, c) \in t_{r,PCB}$
(e) $\{(r',\ p,\ r), (r',\ \texttt{rdf:type},\ c)\} \subseteq \mathcal{B} \Rightarrow (p, in, c) \in t_{r,PCB}$

In other words, each resource is represented by its set of adjacent edges/triples in the graph, with their adjacent nodes/resources being abstracted by their types. Thus, the transaction representing a resource through the PCB conversion is an abstraction of the resource neighborhood. The first motivation for the use of this abstraction is that we are interested in structural patterns at schema level, not by patterns involving specific resources. As a consequence, it can happen that different resources are represented by equivalent transactions. The second motivation for this abstraction is that the complexity of frequent pattern mining approaches in related to the number of unique items appearing in all transactions. Thus, the abstraction is necessary to guarantee the scalability of our measures.

Finally, the PCB representation of an entire RDF graph $\mathcal{B}$ is simply the collection of the PCB representations of all its resources, which results in a database $\mathcal{D}$, i.e., a bag of transactions.

3.2. From RDF Updates to Transactions

In order to evaluate the evolution of RDF graphs, we also need to define a representation of RDF updates in terms of transactions, similar to the PCB representation of RDF graphs. Generically, an update can be seen as a pair of sets of triples $u = \langle Insert, Delete \rangle$. The result of applying update u to $\mathcal{B}$ results in the new RDF graph:

$$\mathcal{B}' = \mathcal{B} + u = \mathcal{B} \setminus Delete \cup Insert$$

Given that *Insert* and *Delete* are sets of triples, like an RDF graph, both could be represented by bags of transactions $\mathcal{D}_{Insert}, \mathcal{D}_{Delete}$. However, this would be a bad representation because the patterns are chosen to compress full descriptions of resources, not deltas of such descriptions. For instance, suppose that an update completes a film description with missing information, e.g., release date and director, and that the chosen patterns represent complete descriptions of films. The description after update will be well compressed because it will contain one of the patterns, but *Insert* will not.

We therefore choose to represent an RDF update by two bags of transactions that represent the state of affected resources respectively before and after the update.

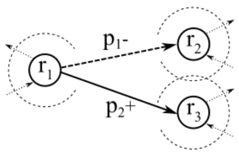
Definition 5. *Let $\mathcal{B}$ be an RDF graph, u be an update, and $AR(u)$ be the set of* affected resources *in u, i.e., the set of resources occuring in u. Its PCB conversion is a tuple $\langle q, q' \rangle$, where $q = \{t_{r,PCB} \mid r \in AR(u)\}$ is the bag of PCB transactions of $AR(u)$ in $\mathcal{B}$ (*before *applying the update), and $q' = \{t'_{r,PCB} \mid r \in AR(u)\}$ is the bag of PCB transactions for $AR(u)$ in $\mathcal{B}' = \mathcal{B} + u$ (*after *the update).*

In the PCB conversion of an update, q is the *initial state* of the affected part of the RDF graph, and q' is its *final state*. Figure 1 shows the localized effect of an update u on an RDF graph. The update is made of one insertion and one deletion, $u = \langle \{(r_1, p_2, r_3)\}, \{(r_1, p_1, r_2)\} \rangle$, and there are three affected resources: $AR(u) = \{r_1, r_2, r_3\}$. The initial state q is defined by $\{t_{r_1,PCB}, t_{r_2,PCB}, t_{r_3,PCB}\}$ and the final state q' by $\{t'_{r_1,PCB}, t'_{r_2,PCB}, t'_{r_3,PCB}\}$. Note that even if the transaction of a resource is left unchanged, we have to include it to assess the structure of the update.

An important remaining issue is about the granularity of RDF updates, i.e., the way modifications are grouped into updates. For example, DBpedia Live[2] publishes updates that affect distant parts of the graph, and are therefore too coarse-grained. On the opposite, updates that would contain a single triple would be too fine-grained. It is desirable that an update combines all insertions and deletions affecting the same resources, and still is as small as possible under this constraint in order to have a fine-grained evaluation of updates. This leads to the following definition of *local update*, i.e., an update that is localized in a small region of interconnected resources.

Definition 6. *Let $u = \langle Insert, Delete \rangle$ be an RDF update. We define the associated set of* local updates *$U_{local}(u)$ as the set of strongly connected components of the undirected graph defined by $Insert \cup Delete$.*

[2]https://wiki.dbpedia.org/online-access/DBpediaLive, last accessed 19th March, 2019.



$$t'_{r1,PCB} = t_{r1,PCB} - \{(p_1,out), (p_1,out,c_{r2})\} + \{(p_2,out), (p_2,out,c_{r3})\}$$
$$t'_{r2,PCB} = t_{r2,PCB} - \{(p_1,in), (p_1,in,c_{r1})\}$$
$$t'_{r3,PCB} = t_{r3,PCB} + \{(p_2,in), (p_2,in,c_{r1})\}$$

Fig. 1. Example of an update with three affected resources.

Finally, we must remark that a *local update* is too small to apply data mining techniques, but not too small to be compressed with different code tables, which is the basis of the newly proposed measures. Otherwise, we could consider it as another RDF graph and use the global comparison method that is presented in the next section.

3.3. Running Example

As a running example, let $\mathcal{O}$ be an ontology/schema which contains three concepts, `Movie (M)`, `Director (D)`, and `Producer (P)`; and three properties, `hasDirector (hD)`, `hasProducer (hP)`, and `releaseDate (rD)`. For illustrative purposes, let $\mathcal{B}_1$ be a very regular knowledge base but incomplete. In $\mathcal{B}_1$, all movies are directed by directors, and have a release date; all directors have directed at least a movie, but the relationships between movies and producers have yet to be added, thus, producers just exist. The code table CT_1 associated to $\mathcal{B}_1$ thus contains at least 3 itemsets related to those desciptions, as shown in Table 2.

Table 2
Code Table CT_1

itemset ID	itemsets
CT_1*-1*	$\{M, (hD, out), (hD, out, D), (rD, out)\}$
CT_1*-2*	$\{D, (hD, in), (hD, in, M)\}$
CT_1*-3*	$\{P\}$
...	...

Let $\mathcal{B}_2$ be the knowledge base $\mathcal{B}_1$ after many updates where the information about the producers of the movies has been almost completely inserted. The associated code table CT_2 contains then longer itemsets, as shown in Table 3. In such a scenario, let us take a resource which has not yet been updated:

```
X a Movie .
X hasDirector Y .
X releaseDate "yyyy-mm-dd" .
Y a Director .
Z a Producer .
```

Table 3
Code Table CT_2

itemset ID	itemsets
CT_2*-1*	$\{M, (hD, out), (hD, out, D), (rD, out), (hP, out), (hP, out, P)\}$
CT_2*-2*	$\{D, (hD, in), (hD, in, M)\}$
CT_2*-3*	$\{P, (hP, in), (hP, in, M)\}$
...	...

whose conversion into PCB representation would be:

`X`: M, (hD, out), (hD, out, D), (rD, out)
`Y`: D, (hD, in), (hD, in, M)
`Z`: P

Taking such a state as starting point, we could update it by inserting a new producer relationship, adding `X hasProducer Z`. The affected resources of this update u would be `X` and `Z`, so q would be as follows:

`X`: M, (hD, out), (hD, out, D), (rD, out)
`Z`: P

and q' as follows:

`X'` : M, (hD, out), (hD, out, D), (rD, out), (hP, out), (hP, out, P)
`Z'` : P, (hP, in), (hP, in, M)

We will use this example to illustrate the update measures presented in the following section.

4. Measuring Structural Similarity with Patterns

In this section, we first recall the definition of the measure we introduced in [8] and then define two new measures about the structural similarity of an RDF graph and an RDF update w.r.t. one or several RDF graphs. In all three measures, we rely on the MDL principle by using code tables as models of the structure of RDF graphs. Along this section, given a knowledge base $\mathcal{B}_i$, $\mathcal{D}_i$ is its transaction-based representation (i.e., database), and $CT_{\mathcal{D}_i}$ the associated code table.

4.1. Through Compression

Following [8], we adopt the similarity of an RDF graph according to another RDF graph as the comparison of the compression ratios achieved by their respective code tables on the first graph. Intuitively, the main idea is that the better one database can be compressed with the code table of another database compared to its own code table, the closer the two databases are structurally. Indeed, this indicates that the two code tables contain similar structural patterns, although they have been obtained independently. Thus, we adopt the following definition of *structural similarity measure*.

Definition 7. *Let $\mathcal{B}_1$ and $\mathcal{B}_2$ be two RDF graphs, and $\mathcal{D}_1$ and $\mathcal{D}_2$ their respective PCB representations. The* structural similarity measure *of $\mathcal{B}_1$ w.r.t. $\mathcal{B}_2$ is defined as:*

$$sim(\mathcal{B}_1|\mathcal{B}_2) = \frac{L(\mathcal{D}_1|CT_{\mathcal{D}_2})}{L(\mathcal{D}_1|CT_{\mathcal{D}_1})}$$

In other words, when comparing $\mathcal{B}_1$ against $\mathcal{B}_2$, the measure compares how well the code table of $\mathcal{D}_2$ is able to compress the transactions in $\mathcal{D}_1$ compared to the compression achieved by the code table of $\mathcal{D}_1$.

A structural similarity close to 1 indicates that the compared RDF graph $\mathcal{B}_1$ is structurally close to the other one ($\mathcal{B}_2$), according to the calculated code tables. Note that this measure exploits the natural asymmetry of the definition to not only measure structural similarity but also check for structural inclusion. Typically, if $sim(\mathcal{B}_1|\mathcal{B}_2) = 1$ but $sim(\mathcal{B}_2|\mathcal{B}_1) > 1$, it implies that $\mathcal{B}_1$ is structurally included in $\mathcal{B}_2$ but not the converse, i.e., $\mathcal{B}_1$ is entirely "explained" by $\mathcal{B}_2$ but $\mathcal{B}_2$ is only partly "explained" by $\mathcal{B}_1$.

4.2. Through Classification

For the comparison of *local updates* to different RDF graphs (e.g., different snapshots of our knowledge base), a difficulty is that the states of local updates are too small for computing a reliable code table that would represent their structures. Indeed, pattern mining techniques require large databases to extract statistically relevant patterns. In this context, we propose to rely on the classification properties of the code tables obtained via MDL principle-based algorithms [9, 15]. According to [9], given a set of code tables CT_i obtained from a set of databases $\mathcal{D}_i$ and a transaction t, if the shortest coding of t is obtained with CT_i, then the prior probability of t being part of a base is maximal for $\mathcal{D}_i$. We extend this notion to RDF graphs and update states as follows.

Definition 8. *Let $\mathcal{B} = \{\mathcal{B}_1, \ldots, \mathcal{B}_n\}$ be a set of n RDF graphs and $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$ the set of their respective representations as transaction databases. Let q be a state of an RDF update. The* structurally closest graph (SCG) *of state q among graphs in $\mathcal{B}$ is defined as:*

$$SCG(q, \mathcal{B}) = argmin_{\mathcal{B}_i \in \mathcal{B}}\ L(q|CT_{\mathcal{D}_i})$$

When an order is defined between the RDF graphs, we can honor it when several of them are detected as the SCG of the same update state. For example, if a temporal order is established in $\mathcal{B}$, in case of a draw, we can further refine the definition by adopting the most recent RDF base as being the SCG. Besides, to avoid mismatches of items appearing in q but not appearing in a code table $CT_{\mathcal{D}_i}$, the sets of items are unified by applying Laplace smoothing to each of the code tables, as suggested in [9]. In brief, it consists in adding 1 to all the usages of the singleton codes associated to the non-witnessed items. In this way (similar as is usually done in Natural Language Processing with *out of vocabulary* words), we can deal with items that were not before in the database.

Running Example. Let us now apply this measure to the running example presented in Section 3.3. When we codify q' using CT_1, we obtain the following:

$$\begin{aligned} L(q'|CT_1) = &\textstyle\sum_{C \in cover(CT_1, X)} L(code_{CT_1}(C)) \\ &+ \textstyle\sum_{C \in cover(CT_1, Z)} L(code_{CT_1}(C)) \end{aligned}$$

where:

$$cover(CT_1, X) = \{CT_1\text{-}1, (hP, out), (hP, out, P)\}$$
$$cover(CT_1, Z) = \{CT_1\text{-}3, (hP, in), (hP, in, M)\}$$

CT_1 cannot cover entirely q' transactions. The items related to producers, which do not appear in CT_1, have to be taken into account to determine the code lengths of our transactions. Thus, we need to rely on the singleton codes of the code table to codify such transaction[3]. On the other hand, when we codify q' using CT_2:

$$\begin{aligned} L(q'|CT_2) = &\textstyle\sum_{C \in cover(CT_2, X)} L(code_{CT_2}(C)) \\ &+ \textstyle\sum_{C \in cover(CT_2, Z)} L(code_{CT_2}(C)) \end{aligned}$$

[3]If such items were not previously observed in the data (and thus, they were not in the code table), Laplace smoothing comes into play, penalizing more such their usage.

where:

$$cover(CT_2, X) = \{CT_2\text{-}1\}$$
$$cover(CT_2, Z) = \{CT_2\text{-}3\}$$

CT_2 covers entirely q' transactions. As D_1 and D_2 contain the same number of transactions and, except for the population of the new property (i.e., `hasProducer`), the corresponding codes of their respective code tables have the same usage (e.g., CT_1-*1* has the same usage as CT_2-*1*), we can assume that the code lengths of each of the codes in the code tables are equal (or really close) to their respective counter parts (e.g., CT_1-*1* has the same code length as CT_2-*1*). Thus, as CT_2 codes cover entirely q' transactions while CT_1 codes have to rely on the sub-optimal codes from the standard code table, the classification according to their codification length would tell us that such update belongs to the most recent version of the knowledge base.

4.3. Through Structural Information Across States

Finally, to check whether a *local update* of an RDF graph respects its current structure, we focus on the difference between the previous and final states. We determine whether the update maintains the current structure of the RDF graph by comparing the quantity of information of the codified transactions in its initial and final states.

Definition 9. *Let $\mathcal{B}$ be an RDF graph, $CT_{\mathcal{B}}$ and $SCT_{\mathcal{B}}$ be the code table and standard code table of its associated database $\mathcal{D}$, respectively. Let u be an update over $\mathcal{B}$, and its PCB representation $\langle q, q' \rangle$, where q is the inital state and q' the final state. We define the* delta of information of *u as:*

$$\delta(u|\mathcal{B}) = \delta_{CT}(u|\mathcal{B}) - L(q \setminus q'|SCT_{\mathcal{B}}) + L(q' \setminus q|SCT_{\mathcal{B}})$$

where

$$\delta_{CT}(u|\mathcal{B}) = L(q|CT_{\mathcal{B}}) - L(q'|CT_{\mathcal{B}})$$

$\delta_{CT}(u|\mathcal{B})$ measures the number of bits that are saved by applying the update. A positive value indicates that the transactions of affected resources are better compressed after the update, and hence they respect better the structure of the RDF graph. A negative value indicates the converse, i.e., a divergence from the structure. $\delta(u|\mathcal{B})$ corrects that measure with two terms that take into account the removed items ($q \setminus q'$) and the added ones ($q' \setminus q$), respectively. Indeed, suppose that a local update maintains all structures in q and adds new items, then we have $\delta_{CT}(u|\mathcal{B}) < 0$ which is counter-intuitive because no structural information was lost, and new information was added. To compensate, we add the cost of representing the additional items using the standard code table, i.e., without taking into account structural information that may exist in the new items. Similarly, an update that would preserve structures except for a few removed items would have $\delta_{CT}(u|\mathcal{B}) > 0$, and we compensate by subtracting the cost of removed items.

By relying on the decomposability of $L(x|SCT_{\mathcal{B}})$ because the standard code table has only singleton patterns, it can be proved that

$$\delta(u|\mathcal{B}) = gain(q'|\mathcal{B}) - gain(q|\mathcal{B})$$

where $gain(x|\mathcal{B}) = L(x|SCT_{\mathcal{B}}) - L(x|CT_{\mathcal{B}})$ measures the gain in bits that we obtain by codifying the bag of transactions x with $CT_{\mathcal{B}}$ instead of with the standard code table. It represents the purely structural information in a state, and $\delta(u|\mathcal{B})$ therefore represents the evolution, positive or negative, of the structural information across states q and q', hence through the update u.

Running example. Let us consider the running example and the values of the delta of information of u. As a reminder, $\mathcal{B}_2$ is the current status of the knowledge base, and q and q' are the initial and final status of the affected transactions, respectively. Thus:

$$\delta(u|\mathcal{B}_2) = \delta_{CT}(u|\mathcal{B}_2) - L(q \setminus q'|SCT_{\mathcal{B}_2}) + L(q' \setminus q|SCT_{\mathcal{B}_2})$$

where:

$$\delta_{CT}(u|\mathcal{B}_2) = L(q|CT_{\mathcal{B}_2}) - L(q'|CT_{\mathcal{B}_2})$$

Unfolding $\delta_{CT}(u|\mathcal{B}_2)$, we have:

$$\begin{aligned}\delta_{CT}(u|\mathcal{B}_2) = \\ & L(code_{CT_2}(M)) + L(code_{CT_2}((hD, out))) \\ & + L(code_{CT_2}((hD, out, D))) + L(code_{CT_2}((rD, out))) \\ & + L(code_{CT_2}(P)) - (L(code_{CT_2}(CT_2\text{-}1)) \\ & + L(code_{CT_2}(CT_2\text{-}3)))\end{aligned}$$

The codifications of the single items are bigger than the codes in the code table, so this delta is going to be positive. Regarding the other two terms in $\delta(u|\mathcal{B}_2)$, given that $q \setminus q' = \emptyset$, we have:

$$L(q \setminus q' | SCT_{\mathcal{B}_2}) = 0$$

On the other hand:

$$\begin{aligned} &L(q' \setminus q | SCT_{\mathcal{B}_2}) = \\ &\quad L(code_{SCT_2}((hP, out))) + L(code_{SCT_2}((hP, out, P))) \\ &\quad + L(code_{SCT_2}((hP, in))) + L(code_{SCT_2}((hP, in, M))) \end{aligned}$$

which is going to be positive as well. Therefore, we would have a positive value for this update as we would be moving the structure of the affected resources closer to the actual structures witnessed in the data. Finally, we could examine the opposite scenario: starting in q', we would have deleted the `hasProducer` triple. In that case, all the previous formulae would have changed completely its signs (i.e., q becomes q' and vice versa) leading to a negative value. Thus, our approach detects if the structure of the update deviate from what became the norm.

5. Data Evolution Assessment

In this section, we propose a method to apply the structural similarity measures presented in Section 3 to assess the evolution of an RDF graph, focusing on different levels of structural conformance. As we can see in Figure 2, our proposal considers two granularity levels.

1. **Global level (snapshots).** Snapshots $\mathcal{S}_i$ are regularly taken as global states of a knowledge base, and define different versions of it. For each snapshot, a code table $CT_{\mathcal{S}_i}$ can be computed, and represents the structural model of the snapshot. We propose to structurally compare the different snapshots in order to assess the evolution of the structural model of the knowledge base as a whole.
2. **Local level (updates).** Between two successive snapshots, there are generally numerous local updates u_i (see Definition 6). Although each local update only affects a few resources, they collectively contribute to the structural evolution of the knowledge base. We propose to assess local updates by comparing their final state q_i' to different snapshots (*state assessment*), as well as by measuring the delta of information between their initial state q_i and final state q_i' w.r.t. the last snapshot (*effects assessment*).

In the remainder of this section, we present in detail the assessment of the evolution of a knowledge base at those two levels.

5.1. Global Assessment of the Evolution

The global level of assessment permits us to structurally compare the global structural model of the knowledge base at the different times chosen by the administrator to define snapshots. The administrator chooses the snapshot times based on the planned evolution of the knowledge base, as well as on the available human and machine resources. This assessment makes it possible to quantify the structural differences between versions and can be used as a signal for potentially breaking evolution of the base data structure for the applications relying on it.

Figure 2 shows the timeline of such an evaluation. Once defined the sequence of snapshots $(\mathcal{S}_1, \ldots, \mathcal{S}_n)$, the structural similarity through compression (Definition 7) can be computed for all pairs of snapshots. Recall that this measure is assymetric. We therefore obtain an assymetric matrix of similarities that can reveal the evolution of the structural model. For example, it can reveal an extension of the vocabulary from snapshot $\mathcal{S}_i$ to snapshot $\mathcal{S}_j$ by showing that the former is included in the latter ($sim(\mathcal{S}_i|\mathcal{S}_j) = 1$), but that the latter is not included in the former ($sim(\mathcal{S}_j|\mathcal{S}_i) > 1$).

From the definition of structural similarity, we can observe that computing $sim(\mathcal{S}_i|\mathcal{S}_j)$ requires both code tables of $\mathcal{S}_i$ and $\mathcal{S}_j$, but only the transaction database of $\mathcal{S}_i$. As a consequence, if we plan to compare future snapshots to the existing snapshots, and not vice versa, we only need to keep the code tables of the existing snapshots, not their transaction databases. We can therefore adopt two versioning strategies depending on the storage restrictions we may have:

- Strong versioning: for each snapshot, we store the code table and the transaction database. Note that the transaction database of a snapshot requires much less space than the original RDF graph.
- Lightweight versioning: for each snapshot, we only store the code table.

5.2. Local Assessment of the Evolution

Being able to detect changes in the global structure of the data is important, but we have not to neglect

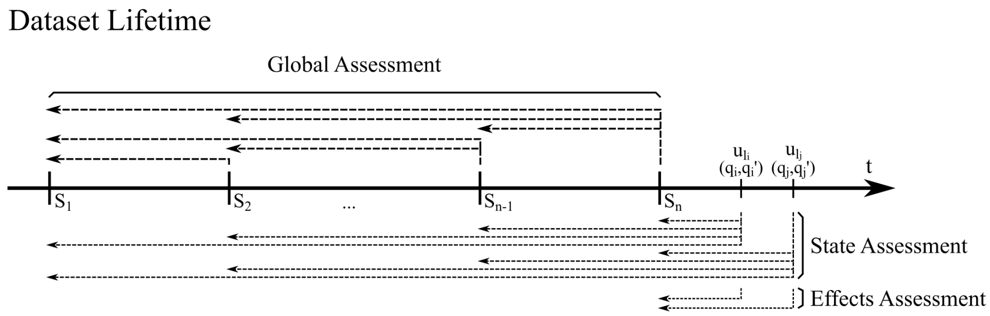


Fig. 2. Different level details proposed for the assessment of the evolution of an RDF base using structural similarities.

the small changes that occur in a more frequent way. In particular, the ability to detect problems at smaller scale would allow to raise detection alarms when the structure of the new updates diverge too much from the structure of the knowledge base. This is specially important in scenarios where information from different sources are integrated (e.g., the results of different automatic extraction tools, or of collaborative edition), as it would make it possible the early detection of discordance among the different sources of data.

At the local level, we propose two different evaluations that rely on the two measures that enable to compare an update to snapshots. They provide information about the impact of local updates on the evolution of the structural model (see Figure 2). **State assessment** allows to check which snapshot of KB a particular update *belongs to* via classification. **Effects assessment** evaluates the effects of an update by itself in terms of increase or decrease in structural conformance.

State Assessment From a set of snapshots, the structural evolution of the data can be checked by determining the closest graph for each of the updates (see Definition 8). In this case, given a local update $u_l = \langle q, q' \rangle$, we will focus on the final state of the data that it modifies (i.e., q').

Thus, given a set of local updates U and a sequence of snapshots $\mathcal{S} = \mathcal{S}_1, \ldots, \mathcal{S}_n$, in order to detect divergences in the structure of the data, we compute the distribution of local updates across snapshots:

$$Distrib_{\mathcal{S}}(U) = \{\langle \mathcal{S}_i, |Class_{\mathcal{S}}(U, \mathcal{S}_i)| \rangle \mid \mathcal{S}_i \in \mathcal{S}\}$$

with for each snapshot $\mathcal{S}_i \in \mathcal{S}$:

$$Class_{\mathcal{S}}(U, \mathcal{S}_i) = \{u = \langle q, q' \rangle \in U \mid SCG(q', \mathcal{S}) = \mathcal{S}_i\}$$

In other words, we count how many final states are classified in each of the previous snapshots of our knowledge base. Ideally, if no schema evolution is expected, all of the updates should be classified in the latest snapshot. If this is not the case, this means that some updates still refer to an outdated snapshot of the knowledge base. If the updates come from different sources, identifying which local updates are outdated enables to identify the problematic data source, and fix it if possible. As an example, in an integration scenario with different ETL pipelines, we could detect outdated versions which are not following the new vocabulary usage guidelines.

Effects Assessment Last but not least, we deal with the most common scenario: only the latest snapshot is given, and we want to evaluate each *local update* to know what is happening with the data as it evolves. The adequacy of a local update u_l w.r.t. a snapshot $\mathcal{S}_i$ is evaluated by measuring the delta of information $\delta(u_l|\mathcal{S}_i)$ (Definition 9). The interpretation of that measure depends on the expectations of the knowledge base administrator:

- A positive value implies that the update modifies the affected data into a state that is structurally closer to the snapshot structures. Thus, in scenarios of data cleaning, this would be a signal of good evolution.
- A negative value implies that the udpate moves away from the snapshot structures. Note that this is not necessarily bad: the data might be evolving for many different reasons. In this case, the administrator has to evaluate whether this change was expected or not by inspecting the concerned local update.

Besides, as this evaluation is built on top of the codification of the updates, the data administrator can find explanations for positive/negative delta values by comparing the patterns that are used in the codification of the initial and final states respectively. For instance, if a pattern is used for the initial state but not for the final state, this indicates that some structure was lost through the update.

6. Experimental Evaluation

In this section, we present the experiments that have been conducted to assess the ability of the proposed measures to detect the evolution of a knowledge base. First, our implemented prototype and the experimental settings of our evaluation are explained. Then, the experiments on synthetic dataset are detailed in order to evaluate the measures in a controlled scenario. Finally, the experiments carried out in a real scenario dataset (DBpedia) are presented to show how our approach can be applied as well as its scalability.

6.1. Experimental Settings

The prototype has been developed in Java 8, using Jena 3.4 to process the RDF data. All the different datasets as well as all the code used in the experiments can be found at http://sid.cps.unizar.es/projects/dataEvolution/. Finally, all the experiments were conducted on a desktop computer with an Intel Core i7-6700K processor (4 cores, 8 threads) at 4.00 GHz and 32 GB of RAM memory.

The code tables are computed thanks to an implementation of SLIM [15][4] which is an any-time version of KRIMP [9].

6.2. Synthetic Experiments

The first experiments were conducted on synthetic data to show the behaviour of our approach in a controlled setting. Firstly, we evaluated the behaviour of the similarity measure (Definition 7) in the presence of very regular datasets, and the influence of their relative size on such a measure. Secondly, we focused on checking whether our proposed delta (Definition 9) captured correctly the artificial alterations of the dataset.

6.2.1. Synthetic Datasets

To generate the synthetic datasets, we used the LUBM data generator [11]. LUBM randomly generates data about universities and their staff, and is commonly used for benchmarking purposes. We

[4]We have used the SLIM2012 Vreeken et al.'s implementation.

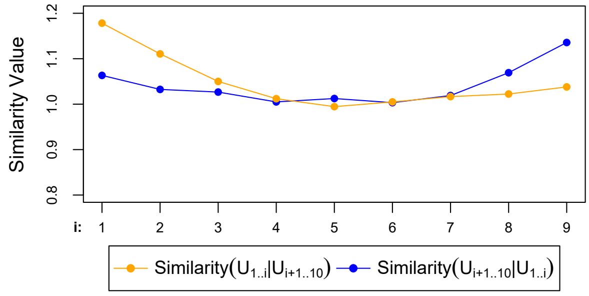


Fig. 3. Structural similarity values between the pairs of merged datasets.

must remark that we have selected LUBM instead of other possible synthetic data generators such as WatDiv [18] or gMark [19] on purpose: LUBM generates very regular datasets which allowed us to isolate, test and validate different properties of the proposed measures. We generated a dataset of 10 universities $\mathcal{U} = \{U_1, \ldots, U_{10}\}$, which contains 1,316,700 RDF triples, and results in 207,433 transactions.

6.2.2. Influence of the Differences of Size Between Datasets

For the first experiment, the 10 universities were considered and their descriptions were merged in order to create graphs of incrementally bigger sizes. To evaluate the structural similarity (Definition 7) under different size ratios, we proceeded as follows.

- We built 9 pairs of merged graphs, $(U_{1..i}, U_{i+1..10})$ with i ranging from 1 to 9, and $U_{i..j} = \bigcup_{k=i}^{j} U_k$.
- For each pair of merged graphs, their respective code tables were computed, and their structural similarity was measured in both directions.

Figure 3 shows the results. The x axis represents the separation index, i.e., the index in the set of universities ($\mathcal{U}$) that is used to split $\mathcal{U}$ into 2 merged datasets. The y axis represents the similarity value. There are two lines (one blue and one orange) because the similarity measure is asymmetric. As expected, all the built datasets are very structurally similar, with structural similarity measures close to 1. By considering their compression ratios (see Table 4)), we also note that they are very regular, as their compression ratios vary from 0.13 for the smallest ones to 0.11 for the biggest ones. However, we can observe the detection of structural differences in the case of extreme size ratios, with the smaller graph being compared to the bigger one.

At first sight, this could be seen as a signal that the relative size actually influences the similarity values, however this increase is due to the fact that the smaller

Table 4

Compression ratios of the merged graphs ($U_{1..i}$, $U_{i+1..10}$).

Pair $U_{1..i}$, $U_{i+1..10}$	Compression Ratios
$U_{1..1}$, $U_{2..10}$	0.135, 0.116
$U_{1..2}$, $U_{3..10}$	0.128, 0.117
$U_{1..3}$, $U_{4..10}$	0.125, 0.117
$U_{1..4}$, $U_{5..10}$	0.122, 0.118
$U_{1..5}$, $U_{6..10}$	0.121, 0.118
$U_{1..6}$, $U_{7..10}$	0.120, 0.120
$U_{1..7}$, $U_{8..10}$	0.118, 0.123
$U_{1..8}$, $U_{9..10}$	0.117, 0.125
$U_{1..9}$, $U_{10..10}$	0.116, 0.130

dataset has less information to grasp during the data mining. This fact is shown as we delve further into the meaning of the measures, to see that Definition 7 can be rewritten as

$$sim(\mathcal{B}_1|\mathcal{B}_2) = \frac{ratio(\mathcal{D}_1|CT_{\mathcal{D}_2})}{ratio(\mathcal{D}_1|CT_{\mathcal{D}_1})}$$

with $ratio(\mathcal{D}_i|CT_{\mathcal{D}_j})$ being the compression ratio achieved by compressing $\mathcal{D}_i$ using the code table obtained for $CT_{\mathcal{D}_j}$. This shows more clearly that the accuracy of the similarity measure does not depend on the relative size of the compared datasets, but on: 1) the presence of structures in $\mathcal{D}_1$ and the ability of the pattern mining technique to obtain them (captured by $ratio(\mathcal{D}_1|CT_{\mathcal{D}_1})$), and 2) the amount of the structural information contained in $CT_{\mathcal{D}_2}$ that is applicable to $\mathcal{D}_1$ (captured by $ratio(\mathcal{D}_1|CT_{\mathcal{D}_2})$). This is coherent with our proposal, as we need enough data in order to calculate proper code tables, and of course, to compare structurally two graphs, some structure must be present in them.

6.2.3. Sensitivity and Polarity of the Delta of Information

This second experiment aims at checking the practical correctness of the *delta of information* (Definition 9), evaluating whether it captures the deviation of structure from the expected usages, and whether it measures such deviation proportionally to its actual divergence. The main idea is to generate updates such that their deviation from the current structure of the KB is known. Thus, starting from the regular dataset, we randomly altered some instances, while keeping track of the modifications. This way, we had two updates for each instance: the *disrupting* one, deviating the state of the instance from the expected state to a potentially structurally wrong; and the *restoring* one, which restored the instance's status to the original one. Our hypothesis was that the disrupting update should have a negative delta as it diverges from the structure, while the restoring update should have a positive delta. Besides, we also wanted to check whether the measured delta for each update actually captures the extent of the modification; to this effect, we also varied the amount of alterations we performed to each instance.

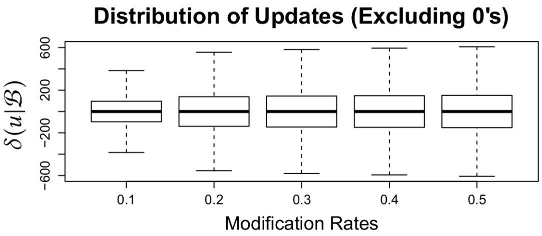


Fig. 4. Distribution of $\delta(u|\mathcal{B})$ for the generated updates.

Thus, we proceeded as follows.

- The code table of the whole dataset was computed to extract the structures[5].
- 25,000 instances were randomly selected (representing ~10% of the instances in the dataset). This set of instances was frozen for the whole experiment in order to alter always the same instances.
- Each selected instance was altered by applying random modifications to some of their triples, ranging from 10% to 50% of them. For each of the selected triples, we randomly applied one of the following modifications: 1) deleting the triple, 2) randomly modifying the property, and 3) randomly modifying the property value. In our setup, the probabilities of applying each one were 0.3 (delete), 0.4 (modify property), and 0.3 (modify object). All modification rates combined, we obtained 125,000 update pairs.

Figure 4 shows the distribution of the delta values over all updates. We can see how the size of the deltas increases as the percentage of alterations is higher, showing that our measure is able to actually capture the divergence from the observed measures. Figure 5 shows the mean values of the absolute values of the deltas for each modification rate. Even though in the boxplot figures it is difficult to appreciate, the values steadily increase with the modifications.

[5]For the complete 10 universities dataset, mining the patterns with SLIM took 20s.

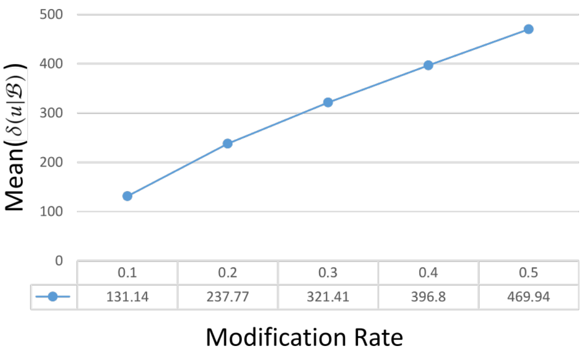


Fig. 5. Means of the absolute values of the deltas.

Finally, we analyzed further the polarity of the delta value of the updates. Ideally, as above mentioned, in a pair of *disrupting-restoring* updates, the former should always be negative, and the latter positive. This happened in 97.67% of the 125 000 update pairs. To discard any potential problem we analyzed the misclassified updates, which were caused by the randomness of the modifications: the amount of misclassified updates decreases as the percentage of modifications increases (1.41% for 0.1, 0.63% for 0.2, and the resting 0.29% was for 0.3 to 0.5 modification rates). While the datasets are really regular, there was still room to have small updates which improved structurally the data; however, the probability of witnessing this behaviour decreases as we increase the disruption, which is coherent with our proposal.

6.3. *Real Scenario Experiments*

To validate our proposal in a real setting we have performed an extensive experimental evaluation over different snapshots of DBpedia [10]. In this section, we first detail the particular experimental settings and implementation. Then, we present the results obtained following the proposal in Section 5. In Section 6.3.2, we assess the evolution of yearly snapshots of DBpedia. In Section 6.3.3, we evaluate the local updates between two periods via classification, and structural information across states.

6.3.1. *DBpedia Dataset and Settings*

To show the feasibility of our proposal in a challenging scenario, we have chosen DBpedia as a source of datasets and updates. On the one hand, the DBpedia project provides different versions, which we used as snapshots in our experiments. On the other hand, the DBpedia Live initiative gives access to intermediate smaller updates, which allowed us to recreate part of its fine-grained evolution. Without loss of generality, we restricted the vocabulary to be included in the snapshots to the DBpedia ontology namespace (i.e., `ontology`, `property` and `resource`). We selected one snapshot per year, starting from DBpedia 3.6. The details of each of the processed snapshots are shown in Table 5[6].

Table 5

Details of the DBpedia snapshots used in the experiments.

Snapshot	#Triples	#Transactions	$\lvert\mathcal{I}\rvert$
3.6 (2011-01)	22,109,064	2,476,538	16,466
3.7 (2011-08)	31,805,464	2,899,989	26,810
3.8 (2012-08)	40,347,137	3,581,783	29,416
3.9 (2013-09)	78,495,071	4,685,189	37,136
2014	106,674,049	5,063,500	45,162
2015-10	100,865,312	5,948,202	61,580
2016-10	102,273,104	6,601,796	61,998

For the DBPedia Live updates, we selected six months between 2015-10 and 2016-10 snapshots (starting in January 2016, the previous months were not available). In total, we considered the 22,495 DBpedia Live updates appearing during those first 6 months of 2016, out of around 900,000 ones applied between the two snapshots. Those 22,495 updates led to 249,619 *local updates* with an average of 2.43 transactions per *local update*.

For the extraction of the code tables, we selected a time threshold for each dataset snapshot that would allow us to obtain comparable compression ratios (and thus, ensure that the structures captured in the code tables contained a similar amount of information about the observed structures in their associated datasets). SLIM ran for 24h on all snapshots except for 2015-10 and 2016-10, for which we had to make it run for 48h in order to reach a similar compression ratio, between 0.260 and 0.300. The obtained compression ratios can be seen in the first column of Table 6.

6.3.2. *Evaluating Dataset Evolution*

The first set of experiments aimed at checking whether the structural measure based on compression was capable to capture strong changes in a real deployment scenario. In Table 6, the measures between the different snapshots of DBpedia, and the associated color map are shown. The first column shows the compression ratio achieved for each snapshot, which repre-

[6] The specific list of files for each snapshot can be found on http://sid.cps.unizar.es/projects/dataEvolution/.

Table 6

Structural similarity through compression between the different snapshots of DBpedia: $sim(DBpedia_{column}|DBpedia_{row})$. The first column shows the compression ratio achieved for each dataset.

	Compression Ratio	DBpedia 3.6	DBpedia 3.7	DBpedia 3.8	DBpedia 3.9	DBpedia 2014	DBpedia 2015-10	DBpedia 2016-10
DBpedia 3.6	0.263	1.0	1.775	1.707	1.895	2.800	2.685	2.450
DBpedia 3.7	0.261	2.851	1.0	1.116	1.346	2.537	2.326	2.150
DBpedia 3.8	0.295	2.996	1.812	1.0	1.221	2.396	2.246	2.108
DBpedia 3.9	0.291	2.910	1.850	1.165	1.0	2.186	2.027	1.914
DBpedia 2014	0.286	3.543	3.468	3.066	3.080	1.0	2.932	2.668
DBpedia 2015-10	0.270	3.254	3.307	2.937	2.980	3.050	1.0	1.261
DBpedia 2016-10	0.306	3.284	3.367	2.993	3.044	3.237	2.395	1.0

sents the amount of structural information that the associated code table captures (the lower, the more information it has). Each cell of other columns contains the measure value, $sim(DBpedia_{column}|DBpedia_{row})$. For example, the last cell of the first row reads as: *the coding length of DBpedia 2016-10 is 2.450 times higher when using the code table of DBpedia 3.6, compared to using the code table of DBpedia 2016-10 itself*. The opposite cell inverses the roles of the two snapshots.

Analysing the matrix of measures, there are clearly two clusters of snapshots ([3.6,3.9] and [2015-10,2016-10]), with snapshot 2014 being apart, as the measures between snapshots in a same cluster are generally lower than between different clusters. Regarding 2014 snapshot, we can see how our approach is able to detect a change compared to the previous versions (which coincided with a major versioning management change in DBpedia) as there is clearly a break in the structure. Focusing on each of the clusters, we can see how the values of the measure generally increase slower row-wise than column-wise when going from older snapshots to newer snapshots. This asymmetry can be explained by the fact that, as the dataset evolves and gets richer, the structural patterns get longer. As a consequence, the transactions of the newer datasets can still be covered by the older patterns, while some transactions of the older datasets can no more be covered by the newer patterns. This is consistent with the observations we presented in [8].

Execution Times In Table 7, we can see the execution times of the different global comparisons performed in these experiments. Recall that the snapshots in the rows are the ones against which we compare the snapshots in the columns (i.e., we codify the transactions contained in the snapshots of the columns). Finally, note that all the comparisons take below 5 minutes, which shows the scalability of our approach.

Broadly speaking, the dominant components of the cost of codification of a database are the number of non-singleton codes that we have to check for each transaction, and the number of transactions[7]. In the current implementation, each measure implies the codification of the database twice[8], once with each code table. To show the tendency of the algorithm, in Figure 6, we plot the execution times against the sum of non-singleton codes in both code tables times the number of transactions. We can see how the execution times follow a linear tendency on this variable ($\#NonSingleton \times \#Transactions$) with a negligible p-value, which makes our approach really scalable.

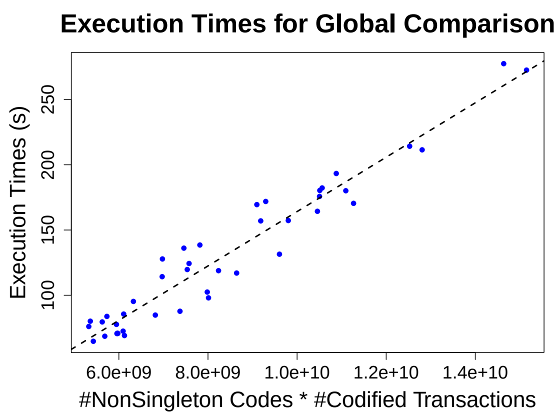


Fig. 6. Execution times as a function of the size of the code table (its non-singleton codes) times the codified transactions.

[7]Note that the actual size of the transaction should be also in the equation, but it is usually much smaller than the other two terms, thus for the tendency we consider it to be a constant.

[8]We acknowledge that there are more efficient ways to calculate the similarity, i.e., storing at least the size of a KB compressed with its own CT.

Table 7
Execution times of the comparisons (in seconds).

	Execution Times (s)						
	DBpedia 3.6	DBpedia 3.7	DBpedia 3.8	DBpedia 3.9	DBpedia 2014	DBpedia 2015-10	DBpedia 2016-10
DBpedia 3.6	-	102.44	117.01	170.44	180.10	277.49	272.55
DBpedia 3.7	84.80	-	87.72	131.46	171.87	214.21	211.44
DBpedia 3.8	70.71	70.87	-	98.00	124.34	175.74	182.24
DBpedia 3.9	70.64	77.64	69.11	-	119.71	164.34	180.310
DBpedia 2014	64.71	76.04	80.12	114.24	-	156.98	169.47
DBpedia 2015-10	72.49	85.57	95.25	118.82	138.49	-	193.31
DBpedia 2016-10	68.58	79.55	83.77	136.11	127.81	157.25	-

Table 8
Details of the DBpedia snapshots and their code tables relevant for execution times.

Snapshot	#NonSingleton	Average Transaction Length
DBpedia 3.6	1,554	16.38
DBpedia 3.7	1,119	22.17
DBpedia 3.8	859	23.24
DBpedia 3.9	851	24.35
DBpedia 2014	637	31.45
DBpedia 2015-10	907	34.77
DBpedia 2016-10	741	31.68

Table 9
Classification of local updates in 2 and 3 DBpedia snapshots.

Class	DBpedia 2014	DBpedia 2015-10	DBpedia 2016-10
in 2 classes	13,834 (5.54%)	235,785 (94.6%)	-
	-	237,707 (95.23%)	11,912 (4.77%)
in 3 classes	13,416 (5.37%)	233,071 (93.37%)	3,132 (1.26%)

This first set of experiments on DBpedia snapshots showed the scalability of the global measure, as well as its capability to be an effective detector of changes between snapshots.

6.3.3. *Evaluating Updates Evolution*

In this batch of experiments, we focused on evaluating the evolution of the data between snapshots. For this, we recreated the evolution of DBpedia in the six first months of 2016 by taking the updates from DBpedia Live, and applying them sequentially. For each of the DBpedia Live update files, we decomposed them into local updates, and computed their associated transactions (see Section 3.2). Then, we applied our proposed evaluation to assess potential structural deviations, and to assess each of the updates individually.

Updates Evolution via Classification As presented in Section 5.2, the code tables of the different snapshots of the knowledge base can be used in order to detect problems in the structure. For instance, when data from different sources are integrated or when the way of producing the data changes. In this set of experiments, the state assessment of the updates is used with the code tables of DBpedia 2014 and 2015-10 to see whether the new updates were globally coherent with the latter structure. Table 9 shows the classification of all the local updates between DBpedia 2014 and 2015-10. For the classification, only the final state of the update is considered. While in a real deployment the latest snapshot would not be available, we also consider DBpedia 2016-10 as a class (second and third rows) to check the actual evolution of the data. Finally, if the codification length of two updates is equal, the classification hit is assigned to the most recent snapshot.

The first two rows of Table 9 present the comparison between two consecutive snapshots. When comparing 2015-10 and 2014 snapshots, most of the updates are classified by the latest code table (2015-10). However, around 5.54% of the updates are still classified as being part of the 2014 snapshot. Using the provenance of the updates, this could be a signal of outdated use of the schema by a particular source. The second row of the table shows how 2016-10 snapshot begins to accept part of the updates (4.77%) even though the proportion of applied updates is still far from being all the changes between the two snapshots (recall that we applied 6 months, 22,495 DBpedia Live updates out of about 900,000 ones, ~2.5% of the total). Finally, the third row of Table 9 presents the results when the three snapshots are taken into account in the analysis. In this last case, 1.26% of the updates are classified as being part of the latest snapshot. We consider this emergent value a good signal, as, while the status of the dataset is still far from 2016-10 snapshot (only ~2.5% of the updates between snapshots were applied), this indicates that the updates are forming structures specific to the next snapshot and our framework is able to detect them.

Single Update Evaluation Finally, we focused on the local evaluation of each of the updates regarding the current structure of the knowledge base. To do so, we performed the *effects assesment* of *local updates* presented in Section 5.2 using the two snapshots they were applied between. Again, we include the 2016-10 code table in the analysis as we want to check whether the measure could detect evolution of the data structure at this level, and which consequences this evolution would have in the resulting snapshot.

Figure 7.a) shows the global distribution of the updates according to their calculated deltas of information ($\delta(u|\mathcal{B})$). We have excluded those updates whose delta was evaluated to zero as they are structurally equivalent, and we have focused on those which actually changed the structure of the affected resources. Recall that positive (negative) values of the delta of an update reflect their increased (decreased) conformance to the observed structures; it is up to the data managers to decide whether they were expecting positive values (e.g., cleaning operation) or negative ones (e.g., change of vocabulary, data evolution, ...). We can see how, in general, the values of the deltas calculated with the 2016-10 snapshot are more compact. This can be a good signal that the evolution of some part of the data has crystallized in the final snapshot.

In order to validate this observation, we focused on what happened to the outliers of 2015-10, and depicted in Figure 7.b) the timeline of such updates (ordered sequentially according their DBpedia Live ID). We can see there how according to 2015-10 we have different punctual peaks of negative values which afterwards are softened in 2016-10. This is a strong evidence of them not being noisy modifications of the structure as their final structures are more relevant in the next snapshot of DBpedia, which is consistent with the classification results presented in the previous experiment. The important remark is that the measure can effectively detect those peaks and provide local information about the structure evolution to raise awareness.

Finally, note that as we are always working with the code tables, the differences in the structure can also be explained exploiting the codifications of the states before and after each update.

7. Related Work

The problem of assessing the evolution of the data in a knowledge base has many different potential approaches. Schema-driven approaches based on the used ontologies, such as Kontokotas et al. [4] and Rieß et al.[5], use a notion of quality based on the respect of explicit rules or patterns, either extracted from the ontology, or stated by an expert. In our approach, such patterns directly emerge from the actual usage of the data. In fact, we consider schema-driven approaches to be complementary to our approach as they give a different view on the evolution: they provide an *a priori* view on what should happen, while our approach allows to assess what has happened and what is happening during the evolution.

The works by Papavasileiou et al. [20], Roussakis et al. [21], and Pernelle et al.[22], while being data-driven evolution approaches as well, have a different goal than ours. Their main focus is on detecting changes and their consequencies on the consistency of the graph. For this, they propose models for the expression of changes in an RDF graph, and provide algorithms to detect high-level changes from low-level ones to infer consequences on the consistency of the graph. In this regard, we have to remark that our detection of changes in data structures using data mining could be directly exploited by Papavasileiou et al. [20]. The closest approach to ours is proposed by González et al. [23]: They use Formal Concept Analysis [24] over RDF datasets to summarize and try to predict the changes in the vocabulary usage. However, the conversion from RDF graph to transaction database that we use captures the structure of the analysed dataset in a more detailed way, and our proposal makes it possible to evaluate the evolution of the dataset at a finer granularity, i.e., at the level of local updates.

Regarding the quality of the updates, Rashid et al. [25] proposed KBQ, a method for the assessment of the quality of updates by focusing on the persistence of resources and their completeness. They provide a more coarse-grained measure of the inner data structure than ours, as they assess quality mainly by counting the number of instances and the number of properties for each class, while we rely on precise emerging patterns. Maillot et al. [26] proposed a method to evaluate the quality of updates of a knowledge base based

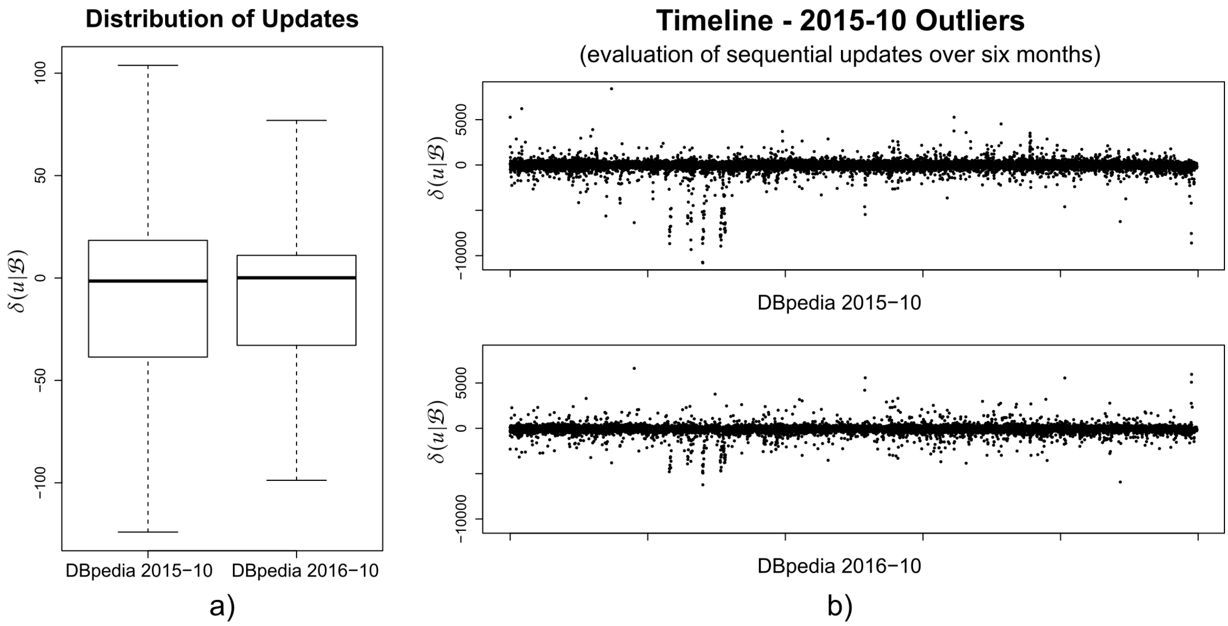


Fig. 7. Information across states of the local updates on DBpedia along six months: a) Distribution of $\delta(u|\mathcal{B})$ values, b) Evaluation of the sequential 2015-10 outlier updates against DBpedia 2015-10 (top), and DBpedia 2016-10 (bottom).

on the Jaccard similarity measure and the generation of relaxed queries. Our proposal handles the updates in a similar way to theirs, but gives more reliable results thanks to the usage of the MDL-based structural similarity measures. Regarding the integration of data sources, Collaran et al. [27] proposed MINTE, a method to merge RDF graphs minimizing redundancy and inconsistencies. To do so, they use a similarity measure based on ontological and graph-edit distance at RDF molecule level. In fact, they could use our different proposed measures to enrich their algorithms. More broadly, our work can be related to the field of datasets dynamics [23, 28, 29] as we give metrics to detect changes in datasets at both snapshot and update level. However, contrary to most approaches in this field, our goal is not to give a detailed resource or schema-level description of the evolution of the base, but to focus on the structural evolution of the graph itself.

Regarding the extracted patterns and their potential usages, we can find that our approach retains the explainability of the differences at the different detail levels, as it is trivial to extract the patterns in a more human-readable format. As an additional benefit, the extracted patterns can also be used to provide a good understanding of the actual inner structure of the dataset. This contrasts with potential approaches based on RDF graph embeddings [17], where global distances between graphs could be defined, but they would not have the structural point of view that our approach provides, and the explainability dimension of the approach would be completely lost. Regarding this aspect, proposals such as LOUPE [30], which helps the users to understand a dataset using the used ontologies and simple triple patterns, could benefit from our extracted patterns. Besides, our extracted patterns could also be used for improving the efficiency of RDF querying. PCB conversion is close to the notion of *characteristic sets* defined by Neumann et al. [31]. Such characteristic sets have been used in various ways to optimize queries over RDF datasets [7, 31, 32]; however, their obtention is not guided by well-established data-mining techniques contrary to our approach.

To the best of our knowledge, no other work have studied the evolution of RDF graphs using data mining techniques at the different granularity levels we propose. Most of the approaches related to the evolution of RDF graphs focus on the detection, description and classification of changes and their impact on the base consistency, while our approach is focused on giving an overview of the structural evolution of the graph.

8. Conclusions and Future Work

In this paper we have addressed the problem of analyzing the evolution of the vocabulary usage in knowledge bases such as RDF graphs. Whereas existing approaches focus on monitoring how the vocabulary usage conforms to a fixed schema, we propose to mine structural patterns contained in data in order to take into account the variability of the usage. Our proposal relies on well established data mining techniques. We have defined two new similarity measures that allow to capture important changes in RDF graphs, both between a knowledge graph and its updates, and between different versions of the same knowledge base, called snapshots. We have also proposed a methodology to apply those measures, providing an assessment tool through the life cycle of the datasets. Last but not least, we have conducted experiments on both synthetic and real datasets (LUBM, DBpedia). The results have shown the ability of our proposal to detect turning points in the evolution of the structure of the data as well as its scalability.

As future work, we plan to further delve into the explainability of the differences between different RDF graphs, and between RDF graphs and the updates applied to them, by exploiting the patterns observed in the data. In addition, the patterns extracted from a knowledge base could be part of the describing metadata of a RDF graph, allowing, for example, to perform comparisons to assess integration of different datasets beforehand.

References

[1] A. Zaveri, A. Rula, A. Maurino, R. Pietrobon, J. Lehmann and S. Auer, Quality assessment for Linked Data: A Survey, *Semantic Web* **7**(1) (2015), 63–93.

[2] M. Ben Ellefi, Z. Bellahsene, J.G. Breslin, E. Demidova, S. Dietze, J. Szymański and K. Todorov, RDF dataset profiling–A survey of features, methods, vocabularies and applications, *Semantic Web* **9**(5) (2018), 1–29.

[3] F. Zablith, G. Antoniou, M. d'Aquin, G. Flouris, H. Kondylakis, E. Motta, D. Plexousakis and M. Sabou, Ontology evolution: A process-centric survey, *The knowledge engineering review* **30**(1) (2015), 45–75.

[4] D. Kontokostas, P. Westphal, S. Auer, S. Hellmann, J. Lehmann, R. Cornelissen and A. Zaveri, Test-driven evaluation of linked data quality, in: *Proc. of Intl. Conf. on World Wide Web (WWW)*, ACM, 2014, pp. 747–758.

[5] C. Rieß, N. Heino, S. Tramp and S. Auer, EvoPat - Pattern-based evolution and refactoring of RDF knowledge bases, in: *Proc. of Intl. Semantic Web Conf. (ISWC)*, Springer, 2010, pp. 647–662.

[6] M.-D. Pham, L. Passing, O. Erling and P. Boncz, Deriving an Emergent Relational Schema from RDF Data, in: *Proc. of Intl. Conf. on World Wide Web (WWW)*, ACM, 2015, pp. 864–874.

[7] M. Meimaris, G. Papastefanatos, N. Mamoulis and I. Anagnostopoulos, Extended characteristic sets: graph indexing for SPARQL query optimization, in: *Proc. of Intl. Conf. on Data Engineering (ICDE)*, IEEE, 2017, pp. 497–508.

[8] P. Maillot and C. Bobed, Measuring Structural Similarity Between RDF Graphs, in: *Proc. of SIGAPP Symposium On Applied Computing (SAC), SWA track*, ACM, 2018, pp. 1960–1967.

[9] J. Vreeken, M. Van Leeuwen and A. Siebes, KRIMP: Mining itemsets that compress, *Data Mining and Knowledge Discovery* **23**(1) (2011), 169–214.

[10] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P.N. Mendes, S. Hellmann, M. Morsey, P. Van Kleef, S. Auer et al., DBpedia–A large-scale, multilingual knowledge base extracted from Wikipedia, *Semantic Web* **6**(2) (2015), 167–195.

[11] Y. Guo, Z. Pan and J. Heflin, LUBM: A benchmark for OWL knowledge base systems, *Web Semantics: Science, Services and Agents on the World Wide Web* **3**(2) (2005), 158–182.

[12] R. Agrawal and R. Srikant, Fast Algorithms for Mining Association Rules in Large Databases, in: *Proc. of Intl. Conf. on Very Large Data Bases (VLDB)*, Morgan Kaufmann Publishers Inc., 1994, pp. 487–499.

[13] M.J. Zaki and C.-j. Hsiao, CHARM: An efficient algorithm for closed itemset mining (2002), 457–473.

[14] T. Uno, T. Asai, Y. Uchida and H. Arimura, An Efficient Algorithm for Enumerating Closed Patterns in Transaction Databases, in: *Discovery Science*, E. Suzuki and S. Arikawa, eds, Lecture Notes in Computer Science, Vol. 3245, Springer Berlin Heidelberg, 2004, pp. 16–31. ISBN 978-3-540-23357-2. http://dx.doi.org/10.1007/978-3-540-30214-8_2.

[15] K. Smets and J. Vreeken, Slim: Directly Mining Descriptive Patterns, in: *Proc. of Intl. Conf. on Data Mining (SDM)*, SIAM/Omnipress, 2012, pp. 236–247.

[16] S. Kramer, N. Lavrač and P. Flach, Propositionalization Approaches to Relational Data Mining, in: *Relational Data Mining*, Springer Berlin Heidelberg, 2001, pp. 262–291.

[17] P. Ristoski, J. Rosati, T. Di Noia, R. De Leone and H. Paulheim, RDF2Vec: RDF graph embeddings and their applications, *Semantic Web* (2018), 1–32.

[18] G. Aluç, O. Hartig, M.T. Özsu and K. Daudjee, Diversified Stress Testing of RDF Data Management Systems, in: *Proc. of Intl. Semantic Web Conf. (ISWC)*, Springer, 2014, pp. 197–212.

[19] G. Bagan, A. Bonifati, R. Ciucanu, G.H.. Fletcher, A. Lemay and N. Advokaat, gMark: Schema-Driven Generation of Graphs and Queries, *IEEE Transactions on Knowledge and Data Engineering* **29**(4) (2017), 856–869.

[20] V. Papavasileiou, G. Flouris, I. Fundulaki, D. Kotzinos and V. Christophides, High-level change detection in RDF (S) KBs, *Trans. on Database Systems (TODS)* **38**(1) (2013), 1.

[21] Y. Roussakis, I. Chrysakis, K. Stefanidis, G. Flouris and Y. Stavrakas, A flexible framework for understanding the dynamics of evolving RDF datasets, in: *Proc. of Intl. Semantic Web Conf. (ISWC)*, Springer, 2015, pp. 495–512.

[22] N. Pernelle, F. Saïs, D. Mercier and S. Thuraisamy, RDF data evolution: efficient detection and semantic representation of changes, in: *Proc. of Intl. Conf. on Semantic Systems (SEMANTiCS) (Posters & Demos)*, CEUR-WS, 2016.

[23] L. González and A. Hogan, Modelling dynamics in semantic web knowledge graphs with formal concept analysis, in: *Proc. of Intl. Conf. on World Wide Web (WWW)*, ACM, 2018, pp. 1175–1184.

[24] B. Ganter and R. Wille, *Formal Concept Analysis: Mathematical Foundations*, Springer-Verlag New York, Inc., 1997.

[25] M. Rashid, G. Rizzo, N. Mihindukulasooriya, M. Torchiano and Ó. Corcho, KBQ - A Tool for Knowledge Base Quality Assessment Using Evolution Analysis, in: *Proc. of Workshops of the Intl. Conf. on Knowledge Capture (K-CAP)*, CEUR-WS, 2017, pp. 58–63.

[26] P. Maillot, T. Raimbault, D. Genest and S. Loiseau, Consistency Evaluation of RDF Data: How Data and Updates are Relevant, in: *Proc. of Intl. Conf. on Signal-Image Technology and Internet-Based Systems (SITIS)*, IEEE, 2014, pp. 187–193.

[27] D. Collarana, M. Galkin, I. Traverso-Ribón, M.-E. Vidal, C. Lange and S. Auer, MINTE: semantically integrating RDF graphs, in: *Proc. of Intl. Conf. on Web Intelligence, Mining and Semantics (WIMS)*, ACM, 2017, pp. 1–11.

[28] J. Umbrich, B. Villazon-Terrazas and M. Hausenblas, Dataset Dynamics Compendium: Where we are so far!, in: *Proc. of COLD at ISWC. Shanghai*, CEUR-WS, 2010.

[29] A. Singh, R. Brennan and D. O'Sullivan, DELTA-LD: A Change Detection Approach for Linked Datasets, in: *Proc. of Workshop on Managing the Evolution and Preservation of the Data Web (MEPDaW)*, CEUR-WS, 2018.

[30] N. Mihindukulasooriya, M. Poveda-Villalón, R. García-Castro and A. Gómez-Pérez, Loupe–An Online Tool for Inspecting Datasets in the Linked Data Cloud, in: *Proc. of Intl. Semantic Web Conf. (Posters & Demos)*, CEUR-WS, 2015.

[31] T. Neumann and G. Moerkotte, Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins, in: *Proc. of Intl. Conf. on Data Engineering (ICDE)*, IEEE, 2011, pp. 984–994.

[32] M.-D. Pham and P. Boncz, Exploiting emergent schemas to make RDF systems more efficient, in: *Proc. of Intl. Semantic Web Conf. (ISWC)*, Springer, 2016, pp. 463–479.

[33] J. Rissanen, Minimum Description Length Principle, in: *Encyclopedia of Statistical Sciences*, John Wiley & Sons, Inc., 2004.

[34] J. Vreeken, M. van Leeuwen and A. Siebes, Characterising the difference, in: *Proc. of SIGKDD Intl. Conf. on Knowledge Discovery and Data mining (KDD)*, ACM, 2007, pp. 765–774.

[35] Q.K. Quboa and M. Saraee, A state-of-the-art survey on Semantic Web mining, *Intelligent Information Management* **5**(1) (2013), 10.

[36] P. Ristoski and H. Paulheim, Semantic Web in data mining and knowledge discovery: A comprehensive survey, *Web Semantics: Science, Services and Agents on the World Wide Web* **36** (2016), 1–22.

[37] V. Nebot and R. Berlanga, Finding association rules in Semantic web data, *Knowledge-Based Systems* **25**(1) (2012), 51–62.

[38] S. Ferré and P. Cellier, Graph-FCA in Practice, in: *Proc. of Intl. Conf. on Conceptual Structures (ICCS'16)*, Springer, 2016, pp. 107–121.

[39] G. Piao and J.G. Breslin, Measuring Semantic Distance for Linked Open Data-enabled Recommender Systems, in: *Proc. of SIGAPP Symposium On Applied Computing (SAC), SWA track*, ACM, 2016, pp. 315–320.

[40] P. Shvaiko and J. Euzenat, Ontology Matching: State of the Art and Future Challenges, *Transact. on Knowledge and Data Engineering* **25**(1) (2013), 158–176, IEEE.

[41] R. Meymandpour and J.G. Davis, A semantic similarity measure for Linked Data: An information content-based approach, *Knowledge-Based Systems* **109** (2016), 276–293.

[42] M. Piernik, D. Brzezinski, T. Morzy and A. Lesniewska, XML clustering: A review of structural approaches, *The Knowledge Engineering Review* **30**(3) (2015), 297–323.

[43] B. Glimm, Y. Kazakov, T. Liebig, T.-K. Tran and V. Vialard, Abstraction Refinement for Ontology Materialization, in: *Proc. of Intl. Semantic Web Conf. (ISWC)*, Springer, 2014, pp. 180–195.

[44] M. Piernik, D. Brzezinski and T. Morzy, Clustering XML documents by patterns, *Knowledge and Information Systems* **46**(1) (2016), 185–212.

[45] H. Paulheim and C. Bizer, Improving the Quality of Linked Data Using Statistical Distributions, *Intl. Journal on Semantic Web and Information Systems (IJSWIS)* **10** (2014), 63–86.

[46] S. Faisal, K.M. Endris, S. Shekarpour, S. Auer and M.E. Vidal, Co-evolution of RDF datasets, in: *Proc. of Intl. Conf. on Web Engineering (ICWE)*, Springer, 2016, pp. 225–243.

[47] T. Käfer, A. Abdelrahman, J. Umbrich, P. O'Byrne and A. Hogan, Observing Linked Data dynamics, in: *Proc. of Extended Semantic Web Conf. (ESWC)*, Springer, 2013, pp. 213–227.

[48] J. Debattista, S. Auer and C. Lange, Luzzu—A Methodology and Framework for Linked Data Quality Assessment, *Journal of Data and Information Quality* **8**(1) (2016), 1–32.

[49] T. Ruan, Y. Li, H. Wang and L. Zhao, From Queriability to Informativity, Assessing "Quality in Use" of DBpedia and YAGO, in: *Proc. of International Semantic Web Conference (ISWC)*, Springer, 2016, pp. 52–68.

[50] J. Han, J. Pei and Y. Yin, Mining Frequent Patterns Without Candidate Generation, in: *Proc. of SIGMOD Intl. Conf. on Management of Data*, ACM, 2000, pp. 1–12.

[51] M.J. Zaki and C.-J. Hsiao, CHARM: An efficient algorithm for closed itemset mining, in: *Proc. of Intl. Conf. on Data Mining (SDM)*, SIAM, 2002, pp. 457–473.

[52] J. Huan, W. Wang and J. Prins, Efficient mining of frequent subgraphs in the presence of isomorphism, in: *Proc. of Intl. Conf. on Data Mining (ICDM)*, IEEE, 2003, pp. 549–552.