Semantic Web 1 (0) 1–5
IOS Press

Towards Fully-fledged Archiving for RDF Datasets

Olivier Pelgrin [a,*], Luis Galárraga [b] and Katja Hose [a]
[a] *Department of Computer Science, Aalborg University, Denmark*
E-mails: olivier@cs.aau.dk, khose@cs.aau.dk
[b] *Inria, France*
E-mail: luis.galarraga@inria.fr

Abstract. The dynamicity of RDF data has motivated the development of solutions for archiving, i.e., the task of storing and querying previous versions of an RDF dataset. Querying the history of a dataset finds applications in data maintenance and analytics. Notwithstanding the value of RDF archiving, the state of the art in this field is under-developed: (i) existing systems are neither scalable nor easy to use, (ii) no solution supports multi-graph RDF datasets, (iii) there is no standard way to query RDF archives, and (iv) solutions do not exploit the evolution patterns of real RDF data. On these grounds, this paper surveys the existing works in RDF archiving in order to characterize the gap between the state of the art and a fully-fledged solution. It also provides *RDFev*, a framework to study the dynamicity of RDF data. We use RDFev to study the evolution of YAGO, DBpedia, and Wikidata, three dynamic and prominent datasets on the Semantic Web. All these insights allow us to set the ground to sketch a fully-fledged archiving solution for RDF data.

Keywords: RDF Archiving, Knowledge Bases

1. Introduction

The amount of RDF data has steadily grown since the conception of the Semantic Web in 2001 [5], as more and more organizations opt for RDF [41] as the format to publish semantic data. For example, by July 2009 the Linked Open Data (LOD) cloud counted on a few more than 90 RDF datasets adding up to almost 6.7B triples [6]. By 2020, these numbers have catapulted to 1200+ datasets[1] and at least 28B triples[2], although estimates based on LODStats [14] suggest more than 10K datasets and 150B+ triples if we consider the datasets with errors omitted by the LOD Cloud [35]. This boom does not only owe credit to the increasing number of data providers, but also to the constant evolution of the datasets in the LOD cloud. This phenomenon is especially true for community-driven initiatives such as DBpedia [3], YAGO [48], or Wikidata [15], and also applies to automatically ever-growing projects such as NELL [9].

Storing and querying the entire edition history of an RDF dataset, a task we call *RDF archiving*, has plenty of applications for data producers and consumers. RDF archives allow data providers, for instance, to study the evolution of the data [18] and track errors for debugging purposes. They can also be of great utility to RDF streaming applications that rely on a structured history of the data [8, 27]. Besides, archives are vital for consumer applications such as data analytics, e.g., mining correction patterns [37, 38] or historical trend analysis [28].

For all the aforementioned reasons, a significant body of literature has started to tackle the problem of RDF archiving. The current state of the art ranges from systems to store and query RDF archives [2, 11, 22, 24, 32, 40, 44, 49, 51], to benchmarks to evaluate such engines [18, 29], as well as temporal extensions for SPARQL [4, 19, 23, 39]. Diverse in architecture and aim, all these works respond to particular use cases. Examples are solutions such as [24, 44] that provide

*Corresponding author. E-mail: olivier@cs.aau.dk.
[1] https://lod-cloud.net/
[2] http://lod-a-lot.lod.labs.vu.nl/

data maintainers with version control management in the spirit of Git. Conversely, other works [22, 39] target data consumers who need to answer time-aware queries such as "obtain the list of house members who sponsored a bill from 2008". In this case the metadata associated to the actual triples is used to answer domain-specific requirements.

Despite this plethora of work, there is currently no available fully-fledged solution for the management of large and dynamic RDF datasets. This situation originates from multiple factors such as (i) the performance and functionality limitations of RDF engines to handle metadata, (ii) the absence of a standard for querying RDF archives, and (iii) a disregard of the actual evolution of real RDF data. This paper elaborates on factors (i) and (ii) through a survey of the state of the art that sheds light on what aspects have not been explored. Factor (iii) is addressed by means of a framework to study the evolution of RDF data applied to three large and ever-changing RDF datasets, namely DBpedia, YAGO, and Wikidata. The idea is to identify the most challenging settings and derive a set of design lessons for fully-fledged RDF archive management. We therefore summarize our contributions as follows:

1. *RDFev*, a metric-based framework to analyze the evolution of RDF datasets
2. A study of the evolution of DBpedia, YAGO, and Wikidata using *RDFev*
3. A detailed survey of existing work on RDF archive management systems and SPARQL temporal extensions
4. An evaluation of Ostrich [49] – a state of the art RDF archiving solution – on the history of DBpedia, YAGO, and Wikidata
5. The sketch of a fully-fledged RDF archiving system that can satisfy the needs not addressed in the literature, as well as a discussion about the challenges in the design and implementation of such a system

This paper is organized as follows. In Section 2 we introduce preliminary concepts. Then, Section 3 presents *RDFev*, addressing contribution (1). Contribution (2) is elaborated in Section 4. In the light of the evolution of real-world RDF data, we then survey the strengths and weaknesses of the different state-of-the-art solutions in Section 5 (contribution 3). Section 6 describes our evaluation on Ostrich [49] addressing contribution 4. The insights from the previous sections are then used to drive the sketch of an optimal RDF archiving system in Section 7, which addresses contribution (5). Finally, Section 8 concludes the paper.

2. Preliminaries

2.1. RDF Graphs

We define an *RDF graph* G as a set of triples $t = \langle s, p, o \rangle$, where $s \in \mathcal{I} \cup \mathcal{B}$, $p \in \mathcal{I}$, and $o \in \mathcal{I} \cup \mathcal{L} \cup \mathcal{B}$ are the subject, predicate, and object of t, respectively. Here, $\mathcal{I}, \mathcal{L}$, and $\mathcal{B}$ are sets of IRIs (entity identifiers), literal values (e.g., strings, integers, dates), and blank nodes (anonymous entities) [41]. The notion of a *graph* is based on the fact that G can be modeled as a directed labeled graph where the predicate p of a triple denotes a directed labeled edge from node s to node o. The RDF W3C standard [41] defines a *named graph* as an RDF graph that has been associated to a label $g \in I$.

2.2. RDF Graph Archives

Intuitively an *RDF graph archive* is a temporally-ordered collection of all the states an RDF graph has gone through since its creation. More formally a graph archive $A = \{G_0, G_1, \ldots, G_{n-1}\}$ is an ordered set of RDF graphs, where each G_i is a *revision* with *revision number* $i \in \mathcal{N}$. A non-initial revision G_i ($i > 0$) is obtained by applying an *update* or *changeset* $u_i = \langle \Delta_i^+, \Delta_i^- \rangle$ to revision G_{i-1}. The sets Δ_i^+, Δ_i^- consist of triples that have been added and deleted respectively from revision G_{i-1} such that $\Delta_i^+ \cup \Delta_i^- \neq \emptyset$ and $\Delta_i^+ \cap \Delta_i^- = \emptyset$. Figure 1 provides a toy RDF graph archive A that models the evolution of the information about the country members of the United Nations (UN) and their diplomatic relationships (*:dr*), i.e., A stores triples such as $\langle$ *:USA*, *a*, *:Country* $\rangle$ or $\langle$ *:USA*, *:dr*, *:Cuba* $\rangle$. The archive consists of two revisions $\{G_0, G_1\}$, where G_1 is obtained by applying update u_1 to the initial revision G_0.

We extend the notion of changesets to arbitrary pairs of revisions i, j with $i < j$, and use the notation $u_{i,j} = \langle \Delta_{i,j}^+, \Delta_{i,j}^- \rangle$. We remark that a graph archive can also be modeled as a collection of 4-tuples $\langle s, p, o, \rho \rangle$, where $\rho \in \mathcal{I}$ is the RDF identifier of revision $i = rv(\rho)$ and $rv \subset \mathcal{I} \times \mathcal{N}$ is a function that maps revision identifiers to natural numbers. We also define the function $ts \subset \mathcal{I} \times \mathcal{N}$ that associates a revision identifier ρ to its commit time, i.e., the timestamp of application of changeset u_i. Solutions for RDF archiving such as [22, 24, 32, 49] implement this logical model in different ways and to different extents. For example, [24, 44] store deltas (and their associated metadata) in additional named graphs using PROV-O [12]. In contrast, [32] stores the temporal metadata in special in-

dexes that optimize for concurrent updates at the expense of temporal consistency, i.e., revision numbers may not be in concordance with the timestamps.

2.3. RDF Dataset Archives

In contrast to an RDF graph archive, an *RDF dataset* is a set $D = \{G^1, G^2, \ldots, G^m\}$ of named graphs where each graph has a label $g^k \in \mathcal{I}$. Differently from revisions in a graph archive, we use the notation G^k for the k-th graph in a dataset, whereas G^k_i denotes the i-th revision of G^k.

The vast majority of existing solutions for RDF archiving can handle the history of a single graph. However, scenarios such as data warehousing [21, 30] may require to keep track of the common evolution of an RDF dataset, for example, by storing the addition and removal timestamps of the different RDF graphs in the dataset. Analogously to the definition of graph archives, we define a *dataset archive* $\mathcal{A} = \{D_0, D_1, \ldots, D_{l-1}\}$ as a temporally ordered collection of RDF datasets. The j-th revision of $\mathcal{A}$ ($j > 1$) can be obtained by applying a *dataset update* U_j to revision D_{j-1}. If D_{j-1} consists of m RDF graphs ($|D_{j-1}| = m$), then the dataset update U_j is a set of at most $m + 1$ changesets: one for each of the graphs modified by U_j, plus a special changeset $\hat{u}_j = \langle \hat{\Delta}^+_j, \hat{\Delta}^-_j \rangle$ ($\hat{\Delta}^+_j \cap \hat{\Delta}^-_j = \emptyset$) that we call the *graph changeset*. The (potentially empty) sets $\hat{\Delta}^+_j, \hat{\Delta}^-_j$ store the labels of the graphs that have been added and deleted in revision j respectively. We enforce the graphs in $\hat{\Delta}^-_j$ to contain no triples. Figure 2 illustrates an example of a dataset archive with two revisions D_0 and D_1. D_0 is a dataset with graphs $\{G^1_0, G^2_0\}$ both at local revision 0. The dataset update U_1 generates a new global dataset revision D_1. U_1 consists of 2 changesets: u^1_1 that updates graph G^1_0 and generates a new local revision G^1_1, and the graph update $\hat{u}_2$ that adds graph G^3 to the dataset and initializes it at revision 0 (denoted by G^3_0). Notice that G^2 remains at revision 0 since it was not modified by the dataset changeset.

As proposed by some RDF engines [20, 47], we define the master graph G^M (with label M) as the RDF graph that stores the metadata about all the graphs in an RDF dataset. If we associate the creation of a graph G^k with label g^k to a triple of the form $\langle g^k, \textit{rdf:type}, \eta\textit{:Graph} \rangle$ in G^M for some namespace η, then we can model a dataset archive as a set of 5-tuples $\langle s, p, o, \rho, \zeta \rangle$. Here, $\rho \in \mathcal{I}$ is the RDF identifier of the local revision of the triple in an RDF graph with label $l(\rho) = g$. The function $l(\cdot)$ is defined so that it returns the graph label associated to local revision ρ. Conversely, $\zeta \in \mathcal{I}$ identifies a (global) dataset revision $j = rv(\zeta)$. We also overload the function $ts(\zeta)$ defined in Section 2.2 so that it returns the timestamp associated to the dataset revision identifier ζ. Last, we notice that the addition of a non-empty graph to a dataset archive generates two revisions: one for creating the graph, and one for populating it. A similar logic applies to graph deletion.

2.4. SPARQL

SPARQL 1.1 is the W3C standard language to query RDF data [45]. For the sake of brevity, we do not provide a rigorous definition of the syntax and semantics of SPARQL queries; instead we briefly introduce the syntax of a subset of SELECT queries and refer the reader to the official specification [45]. SPARQL is a graph-based language whose building blocks are triple patterns. A *triple pattern* $\hat{t}$ is a triple $\langle \hat{s}, \hat{p}, \hat{o} \rangle \in (\mathcal{I} \cup \mathcal{B} \cup \mathcal{V}) \times (\mathcal{I} \cup \mathcal{V}) \times (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L} \cup \mathcal{V})$, where $\mathcal{V}$ is a set of variables such that $(\mathcal{I} \cup \mathcal{B} \cup \mathcal{L}) \cap \mathcal{V} = \emptyset$ (variables are always prefixed with the symbol ?). A *basic graph pattern* (BGP) $\hat{G}$ is the conjunction of a set of triple patterns $\{ \hat{t}_1 \,.\, \hat{t}_2 \;\ldots\; .\, \hat{t}_m \}$, e.g.,

$$\{ \textit{ ?s a :Person . ?s :nationality :France } \}$$

When no named graph is specified, the SPARQL standard assumes that the BGP is matched against a *default graph* defined by the SPARQL engine. Otherwise, for matches against specific graphs, SPARQL supports the syntax *GRAPH* $\bar{g}$ $\{\hat{G}\}$, where $\bar{g} \in \mathcal{I} \cup \mathcal{V}$. In this paper we call this, a *named BGP* denoted by $\hat{G}_{\bar{g}}$. A SPARQL select query Q on an RDF dataset has the basic form "SELECT V (FROM NAMED $\bar{g}_1$ FROM NAMED $\bar{g}_2 \ldots$) WHERE $\{\hat{G}'\ \hat{G}''\ \ldots\ \hat{G}_{\bar{g}_1}\ \hat{G}_{\bar{g}_2} \ldots\}$, with projection variables $V \subset \mathcal{V}$. SPARQL supports named BGPs $\hat{G}_{\bar{g}}$ with variables $\bar{g} \in \mathcal{V}$. In most settings the bindings for those variables originate from the master graph G_M. The BGPs in the expression can contain FILTER conditions, be surrounded by OPTIONAL clauses, and be combined by means of UNION clauses.

2.5. Queries on Archives

Queries on graph/dataset archives may combine results coming from different revisions in the history of the data collection in order to answer an information need. The literature defines five types of queries on RDF archives [18, 49]. We illustrate them by means of our example graph archive from Figure 1.

G_0	u_1	G_1
$\langle$:*USA*, *a*, :*Country*$\rangle$	$\Delta_1^+ = \{\langle$:*France*, *a*, :*Country*$\rangle\}$	$\langle$:*USA*, *a*, :*Country*$\rangle$
$\langle$:*Cuba*, *a*, :*Country*$\rangle$	$\Delta_1^- = \{\langle$: *USA*, : *dr*, : *Cuba*$\rangle\}\}$	$\langle$:*Cuba*, *a*, :*Country*$\rangle$
$\langle$:*USA*, :*dr*, :*USA*$\rangle$		$\langle$:*France*, : *a*, :*Country*$\rangle$

Fig. 1. Two revisions G_0, G_0 and a changeset u_1 of an RDF graph archive A

D_0	U_1	D_1
$G_0^1 = \{\langle$:*USA*, *a*, :*Country*$\rangle$,	$u_1^1 = \{\Delta_1^{1+} = \{\langle$:*France*, *a*, :*Country*$\rangle\}$,	$G_1^1 = \{\langle$:*USA*, *a*, :*Country*$\rangle$
$\langle$:*Cuba*, *a*, :*Country*$\rangle$,	$\Delta_1^{1-} = \{\langle$: *USA*, : *dr*, : *Cuba*$\rangle\}\}$	$\langle$:*Cuba*, *a*, :*Country*$\rangle$
$\langle$:*USA*, : *dr*, :*USA*$\rangle\}$	$\hat{u}_2 = \{\hat{\Delta}_2^+ = \{G^3\},\ \hat{\Delta}_2^- = \emptyset\}$	$\langle$:*France*, : *a*, :*Country*$\rangle\}$
$G_0^2 = \{\langle x$:*JFK*, *a*, *x*:*Airport*$\rangle\}$		$G_0^2 = \{\langle x$:*JFK*, *a*, *x*:*Airport*$\rangle\}$
		$G_0^3 = \emptyset$

Fig. 2. A dataset archive $\mathcal{A}$ with two revisions D_1, D_2. The first revision contains two graphs G^1 and G^2. The dataset update $\hat{U}_1$ (i) modifies G^1, creating a new local revision G_1^1, (ii) and creates a new graph G^3

– **Version Materialization.** VM queries are standard queries run against a single revision, such as *what was the list of countries according to the UN at revision j?*
– **Delta Materialization.** DM queries are standard queries defined on a set of changes $u_j = \langle \Delta_j^+, \Delta_j^- \rangle$, e.g., *which countries were added to the list at revision j?*
– **Version.** V queries ask for the revisions where a particular query yields results. An example of a V query is: *in which revisions j did USA and Cuba have diplomatic relationships?*
– **Cross-version.** CV queries result from the combination (e.g., via joins, unions, aggregations, differences, etc.) of the information from multiple revisions, e.g., *which of the current countries was not in the original list of UN members?*
– **Cross-delta.** CD queries result from the combination of the information from multiple sets of changes, e.g., *what are the revisions j with the largest number of UN member adhesions?*

Existing solutions differ in the types of queries they support. For example, Ostrich [49] allows for queries of types VM, DM, and V on single triple patterns, whereas [2] supports VM, V, and CV queries on arbitrary BGPs. Even though our examples use the revision number $j = rv(\rho)$ to identify a revision, some solutions may directly use the revision identifier ρ or the revision's commit time $ts(\rho)$. This depends on the system's data model.

3. Framework for the Evolution RDF Data

This section proposes *RDFev*, a framework to understand the evolution of RDF data. The framework consists of a set of metrics and a software tool to calculate those metrics throughout the history of the data. The metrics quantify the changes between two revisions of an RDF graph or dataset and can be categorized into two families: metrics for low-level changes, and metrics for high-level changes. Existing benchmarks, such as [18], focus on low-level changes, that is, additions and deletions of triples. This, however, may be of limited use to data maintainers, who may need to know the semantics of those changes, for instance, to understand whether additions are creating new entities or editing existing ones. On these grounds, we propose to quantify changes at the level of entities and object values (which we call high-level).

RDFev takes as input an ordered set of revisions as RDF dumps in N-triples format and computes the different metrics for each consecutive pair of revisions. *RDFev* is implemented in C++ and Python and uses the RocksBD[3] key-value store as storage and indexing

[3] http://rocksdb.org

backend. For the time being RDFev does not support multi-graph RDF datasets. Its source code is available on https://www.dropbox.com/s/c3j0sce9tuodmyn/rdfev.tar.gz.

3.1. Low-level Changes

Low-level changes are changes at the triple level[4]. Indicators for low-level changes focus on additions and deletions of triples and vocabulary elements. The vocabulary $\Upsilon(D) \subset \mathcal{I} \cup \mathcal{L} \cup \mathcal{B}$ of an RDF dataset D is the set of all the terms occurring in triples of the dataset. Tracking changes in the number of triples rather than in the raw size of the RDF dumps is more informative for data analytics, as the latter option is sensitive to the chosen serialization format. Moreover an increase in the vocabulary of a dataset can provide hints about the nature of the changes and the novelty of the data incorporated in a new revision. All metrics are defined in [18] for pairs of revisions i, j with $j > i$.

Change ratio. The authors of BEAR [18] define the change ratio between two revisions i and j as

$$\delta_{i,j} = \frac{|\Delta_{i,j}|}{|D_i \cup D_j|} \tag{1}$$

$\delta_{i,j}$ compares the size of the changes between two revisions w.r.t. the revisions' joint size. Large values for $\delta_{i,j}$ denote important changes in between the revisions. For a more fine-grained analysis [18] also proposes the insertion and deletion ratios:

$$\delta^+_{i,j} = \frac{\Delta^+_{i,j}}{|D_i|} \tag{2} \qquad \delta^-_{i,j} = \frac{\Delta^-_{i,j}}{|D_i|} \tag{3}$$

Vocabulary dynamicity. The vocabulary dynamicity for two revisions i and j is defined as [18]:

$$\text{vdyn}_{i,j} = \frac{|\Upsilon(\Delta_{i,j})|}{|\Upsilon(D_i) \cup \Upsilon(D_j)|} \tag{4}$$

$\Upsilon(\Delta_{i,j})$ is the set of vocabulary terms—IRIs, literals, or blank nodes— in the changeset $\Delta_{i,j}$. The literature also defines the vocabulary dynamicity for insertions (vdyn+$_{i,j}$) and deletions (vdyn-$_{i,j}$):

$$\text{vdyn+}_{i,j} = \frac{|\Upsilon(\Delta^+_{i,j})|}{|\Upsilon(D_i) \cup \Upsilon(D_j)|} \tag{5}$$

$$\text{vdyn-}_{i,j} = \frac{|\Upsilon(\Delta^-_{i,j})|}{|\Upsilon(D_i) \cup \Upsilon(D_j)|} \tag{6}$$

Growth ratio. The grow ratio is the ratio between between the number of triples in two revisions i, j. It is calculated as follows:

$$\Gamma_{i,j} = \frac{|D_j|}{|D_i|} \tag{7}$$

3.2. High-level Changes

A high-level change confers semantics to a changeset. For example, if an update consists of triples about an unseen subject, we can interpret the triples as the addition of an entity to the dataset. High-level changes provide deeper insights about the development of an RDF dataset than low-level changes. In addition, they can be domain-dependent. Some approaches [36, 43] have proposed vocabularies to describe changesets in RDF data as high-level changes. Since our approach is oblivious to the domain of the data, we propose a set of metrics on domain-agnostic high-level changes.

Entity changes. RDF datasets describe real-world entities s by means of triples $\langle s, p, o\rangle$. Hence, an entity is a subject for the sake of this analysis. We define the metric *entity changes* between revisions i, j as $\sigma_{i,j} = \sigma^+_{i,j} \cup \sigma^-_{i,j}$, where $\sigma^+_{i,j}$ is the set of added entities, i.e., the subjects present in $\Upsilon(D_j)$ but not in $\Upsilon(D_i)$ (analogously the set of deleted entities $\sigma^-_{i,j}$ is defined by swapping the roles of i and j). We also propose the *triple-to-entity-change score*, that is, the average number of triples that constitute a single entity change. It can be calculated as follows:

$$\text{ect}_{i,j} = \frac{|\langle s, p, o\rangle \in \Delta_{i,j} : s \in \sigma_{i,j}|}{|\sigma_{i,j}|} \tag{8}$$

Object Updates and Orphan Object Addition/Deletions. An object update in a changeset $\Delta_{i,j}$ is defined by the deletion of a triple $\langle s, p, o\rangle$ and the addition of a triple $\langle s, p, o'\rangle$ with $o \neq o'$. Once a triple in a changeset has been assigned to a high-level change, the triple is *con-*

[4] All these metrics can easily be ported to quads, however we focus on triples since our experimental datasets are provided as single RDF graphs.

sumed and cannot be assigned to any other high-level change. We define orphan object additions and deletions respectively as those triples $\langle s^+, p^+, o^+ \rangle \in \Delta^+_{i,j}$ and $\langle s^-, p^-, o^- \rangle \in \Delta^-_{i,j}$ that has not been consumed by any of the previous high-level changes.

4. Evolution Analysis of RDF Datasets

Having introduced *RDFev*, we use it to conduct an analysis of the revision history of three large and publicly available RDF knowledge bases, namely YAGO, DBpedia, and Wikidata. The analysis resorts to the metrics defined in Sections 3.1 and 3.2 for every pair of consecutive revisions.

4.1. Data

We chose the YAGO [48], DBpedia [3], and Wikidata [15] knowledge bases for our analysis, because of their large size, dynamicity, and central role in the Linked Open Data initiative. We build an RDF graph archive by considering each release of the knowledge base as a revision. None of the datasets is provided as a monolithic file, instead, they are divided into *themes*. These are subsets of triples of the same nature, e.g., triples with literal objects extracted with certain extraction method. We thus focus on the most popular themes. For DBpedia we use the *mapping-based objects* and *mapping-based literals* themes, which are available from version 3.5 (2015) onwards. Additionally, we include the *instance-types* theme as well as the *ontology*. For YAGO, we use the knowledge base's core, namely, the themes *facts*, *meta facts*, *literal facts*, *date facts*, and *labels* available from version 2 (v.1.0 was not published in RDF). As for Wikidata, we use the *simple-statements* of the RDF Exports [1][5]. Table 1 maps revision numbers to releases for the sake of conciseness in the evolution analysis.

4.2. Low-level Evolution Analysis

Change ratio. Figures 3a, 3d and 3g depict the evolution of the change, insertion, and deletion ratios for our experimental datasets. Up to v.3.9 (rev. 5), DBpedia exhibits a steady growth with significantly more insertions than deletions. Minor versions such as 3.5.1 (rev. 1) are indeed minor in terms of low-level changes.

[5] http://tools.wmflabs.org/wikidata-exports/rdf/index.html

Revision	DBpedia	YAGO	Wikidata
0	3.5	2s	2014-05-26
1	3.5.1	3.0.0	2014-08-04
2	3.6	3.0.1	2014-11-10
3	3.7	3.0.2	2015-02-23
4	3.8	3.1	2015-06-01
5	3.9		2015-08-17
6	2015-04		2015-10-12
7	2015-10		2015-12-28
8	2016-04		2016-03-28
9	2016-10		2016-06-21
10	2019-08		2016-08-01

Table 1
Datasets revision mapping

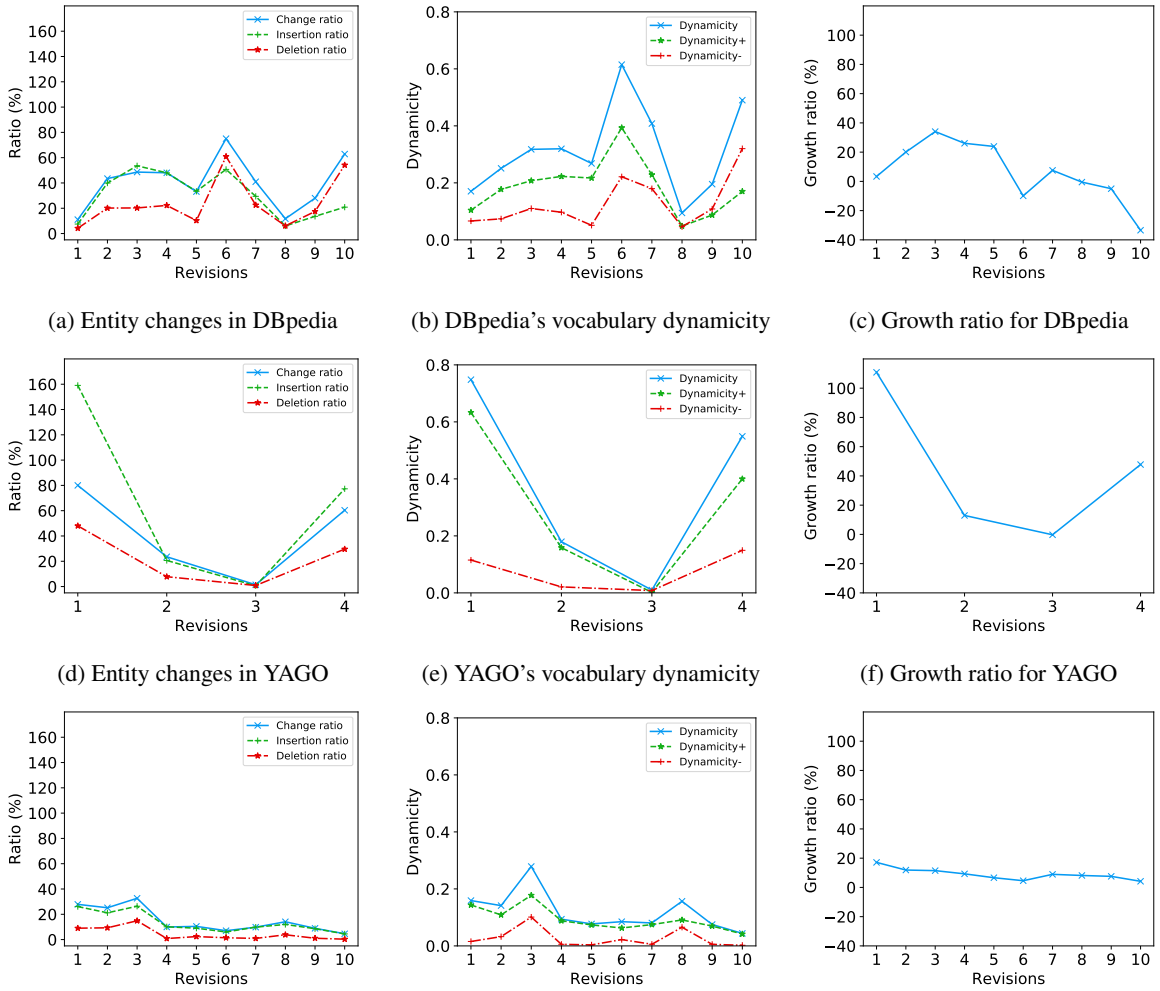
Version 2015-04 (rev. 6) is an inflexion point not only in terms of naming scheme (see Table 1): the deletion rate exceeds the insertion rate and subsequent revisions exhibit a tight difference between the rates. This suggests a major design shift in the construction of DBpedia from revision 6.

As for YAGO, the evolution reflects a different release cycle. There is a clear distinction between major releases (3.0.0 and 3.1, i.e., rev. 1 and 2) and minor releases (3.0.1 and 3.0.2, i.e., rev. 2 and 3). The magnitude of the changes in major releases is significantly higher for YAGO than for any DBpedia release. Minor versions seem to be mostly focused on corrections, with a low number of changes.

Contrary to the other datasets, Wikidata shows a slowly decreasing change rate that fluctuates between 5% (rev. 10) and 33% (rev. 3) within a period of 2 years.

Vocabulary dynamicity. As shown in Figures 3b, 3e, and 3h, the vocabulary dynamicity is, not surprisingly, correlated with the change ratio. Nevertheless, the vocabulary dynamicity between v.3.9 and v.2015-14 (rev. 5 and 6) in DBpedia did not decrease. This means that DBpedia 2015-14 contained more entities, but fewer – potentially noisy – triples about those entities. The major releases of YAGO (rev. 1 and 4) show a notably higher vocabulary dynamicity than the minor releases. As for Wikidata, slight spikes in dynamicity can be observed at revisions 4 and 9, however this metric remains relatively low in Wikidata compared to the others bases.

Growth ratio. Figures 3c, 3f, and 3i depict the growth ratio of our experimental datasets. In all cases, this metric is mainly positive with low values for minor re-



(a) Entity changes in DBpedia (b) DBpedia's vocabulary dynamicity (c) Growth ratio for DBpedia

(d) Entity changes in YAGO (e) YAGO's vocabulary dynamicity (f) Growth ratio for YAGO

(g) Entity changes in Wikidata (h) Wikidata's vocabulary dynamicity (i) Growth ratio for Wikidata

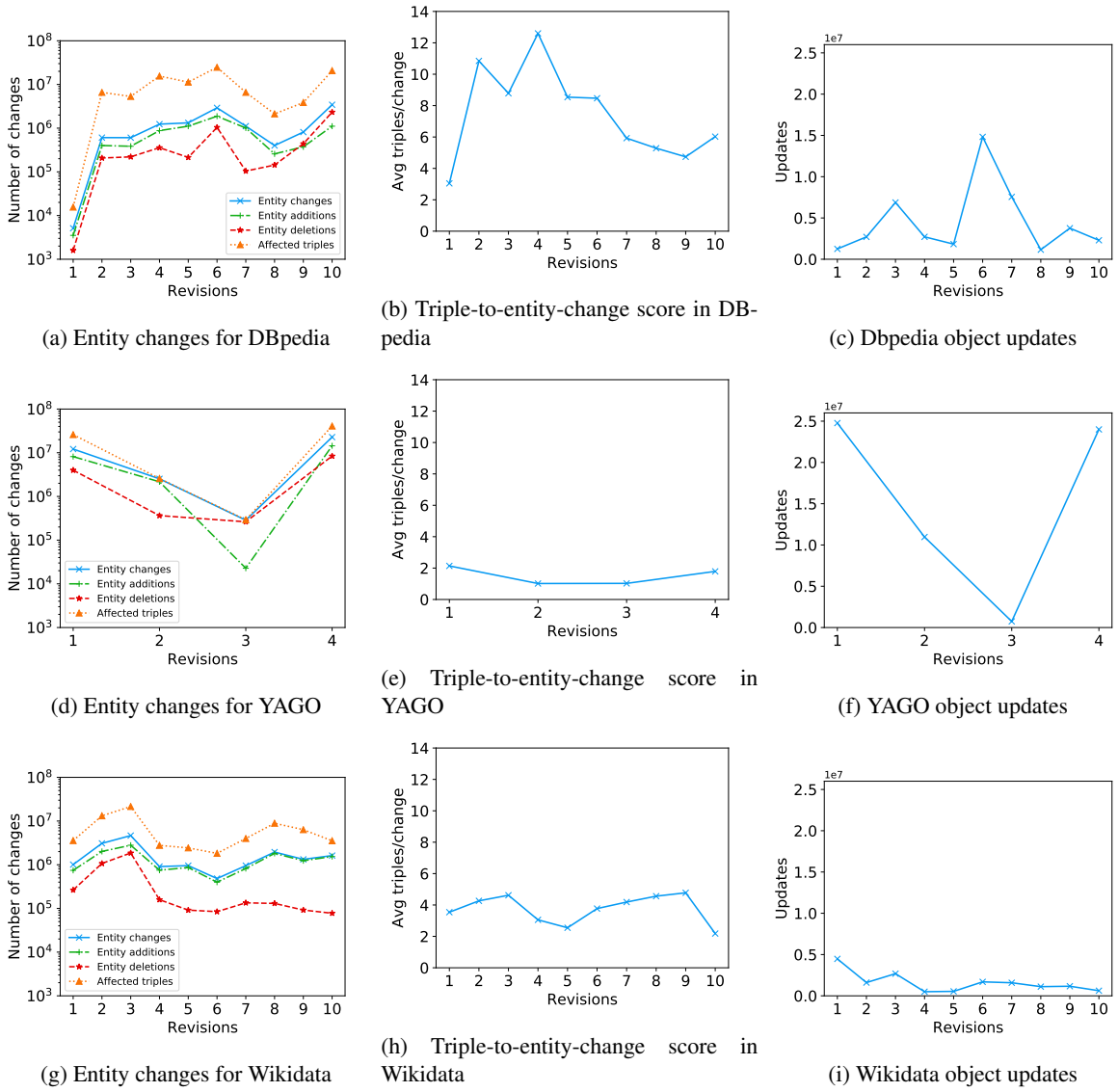
Fig. 3. Change ratio, vocabulary dynamicity, and growth ratio

visions. As pointed out by the change ratio, v.2015-04 in DBpedia is remarkable as the dataset shrank and was succeeded by more conservative growth ratios. This may suggest that recent DBpedia releases are more curated. We observe that YAGO's growth ratio is significantly larger for major versions. This is especially true for the 3.0.0 (rev. 1) release that doubled the size of the knowledge base.

4.3. High-level Evolution Analysis

4.3.1. Entity changes

Figures 4a, 4d, and 4g illustrate the evolution of the entity changes, additions, and deletions for DBpedia, YAGO and Wikidata. We also show the number of triples used to define these high-level changes (labeled as *affected triples*). We observe a stable behavior for these metrics in DBpedia except for the minor release 3.5.1 (rev. 1). Entity changes in Wikidata also display a monotonic behavior, even though the deletion rate tends to decrease from rev. 4. In YAGO, the number of entity changes peaks for the major revisions (rev. 1 and 4), and is one order of magnitude larger than for minor revisions. The minor release 3.0.2 (rev. 3) shows the lowest number of additions, whereas deletions remain stable w.r.t release 3.0.1 (rev. 2). This suggests that these two minor revisions focused on improving the information extraction process, which removed a large number of noisy entities.



(a) Entity changes for DBpedia

(b) Triple-to-entity-change score in DBpedia

(c) Dbpedia object updates

(d) Entity changes for YAGO

(e) Triple-to-entity-change score in YAGO

(f) YAGO object updates

(g) Entity changes for Wikidata

(h) Triple-to-entity-change score in Wikidata

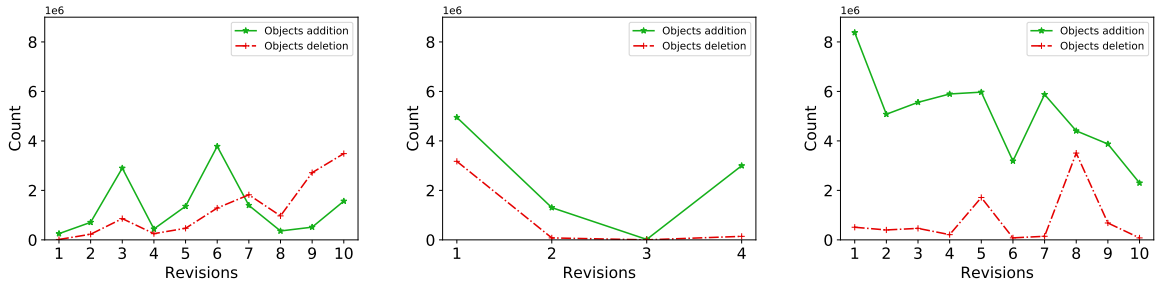
(i) Wikidata object updates

Fig. 4. Entity changes and object updates

Figure 4b shows the triple-to-entity-change score in DBpedia. Before v.2015-14 this metric fluctuates between 2 and 12 triples without any apparent pattern. Conversely subsequent releases show a decline, which suggests a change in the extraction strategies for the descriptions of entities. The same cannot be said about YAGO and Wikidata (Figures 4e and 4h), where values for this metric are significantly lower than for DBpedia, and remain almost constant. This shows that minor releases in YAGO improved the strategy to extract entities, but did not change much the amount of extracted triples per entity.

4.3.2. *Object Updates and Orphan Object Additions/Deletions*

We present the evolution of the number of object updates for our experimental datasets in Figures 4c, 4f, and 4i. For DBpedia, the curve is consistent with the change ratio (Figure 3a). In addition to a drop in size, v.2015-04 also shows the highest number of object updates, which corroborates the presence of a drastic redesign of the dataset. The results for YAGO are depicted in Figure 4f, where we see larger numbers of object updates compared to major releases in DBpedia. This is consistent with the previous results that show



(a) Dbpedia (b) YAGO (c) Wikidata

Fig. 5. Orphan object additions and deletions

that YAGO goes through bigger changes between releases. The same trends are observed for the number of orphan object additions and deletions in Figures 5a and 5b. Compared to the other two datasets, Wikidata's number of object updates, shown in Figure 4i, is much lower and constant throughout the stream of revisions.

Finally we remark that in YAGO and DBpedia, object updates are 4.8 and 1.8 times more frequent than orphan additions and deletions. This entails that the bulk of editions in these knowledge bases aims at updating existing object values. This behavior contrasts with Wikidata, where orphan object updates are 3.7 times more common than proper object updates. As depicted in Figure 5c, Wikidata exhibits many more orphan object updates than the other knowledge bases. Moreover, orphan object additions are 19 times more common than orphan object deletions.

4.4. Conclusion

In this section we have conducted a study of the evolution of three large RDF knowledge bases using our proposed framework *RDFev*, which resorts to a domain-agnostic analysis from two perspectives: At the low-level it studies the dynamics of triples and vocabulary terms across different versions of an RDF dataset, whereas at the high-level it measures how those low-level changes translate into updates to the entities described in the experimental datasets. All in all, we have identified two patterns of evolution. On the one hand, Wikidata exhibits a stable release cycle as our metrics did not exhibit big fluctuations from release to release. On the other hand, YAGO and DBpedia have a release cycle that distinguishes between minor and major releases. Major releases are characterized by a large number of updates in the knowledge base and may not necessarily increase its size. Conversely, minor releases incur in at least one order of magnitude fewer changes than major releases and seem to focus on improving the quality of the knowledge base, for instance, by being more conservative in the number of triple and entity additions. We argue that an effective solution for large-scale RDF archiving should be able to adapt to at least these two patterns of evolution.

5. Survey of RDF Archiving Solutions

We structure this section in three parts. Section 5.1 surveys the existing engines for RDF archiving and discusses their strengths and weaknesses. Section 5.2 presents the languages and SPARQL extensions to express queries on RDF archives. Finally, Section 5.3 introduces various endeavors on analysis and benchmarking of RDF archives.

5.1. RDF Archiving Systems

There are plenty of systems to store and query the history of an RDF dataset. Except for a few approaches [2, 24, 51], most available systems support archiving of a single RDF graph. Ostrich [49], for instance, manages quads of the form $\langle s, p, o, i \rangle$ with revision number $i = rv(\rho)$. Other solutions [22] do not support revision numbers and use the ρ-component $\rho \in \mathcal{I}$ to model temporal metadata such as insertion, deletion, and validity timestamps for triples. In this paper we make a distinction between insertion/deletion timestamps for triples and validity intervals. While the former are unlikely to change, the latter are subject to modifications because they constitute domain information, e.g., the validity of a marriage statement. This is why the general data model introduced in Sec-

tion 2 only associates revision numbers and commit timestamps to the fourth component ρ, whereas other types of metadata are still attached to the graph label g. We summarize the architectural spectrum of RDF archiving systems in Table 2 where we characterize the state-of-the-art approaches according to the following criteria:

- **Storage paradigm.** The storage architecture is probably the most important feature as it shapes the system's architecture. We identify three main storage paradigms in the literature [17], namely Independent Copies (IC), Change-based (CB), and Timestamp-based (TB).
- **Data model.** It can be quads or 5-tuples with different semantics for the fourth and fifth component.
- **Full BGPs.** This feature determines whether the system supports queries with arbitrary BGPs or single triple patterns.
- **Query types.** This criterion lists the types of queries on RDF archives (see Section 2.5) natively supported by the solution.
- **Branch & tags.** It defines whether the system supports branching and tagging as in classical version control systems.
- **Multi-graph.** This feature determines if the system supports archiving of the history of multi-graph RDF datasets.
- **Concurrent updates.** This criterion determines whether the system supports concurrent updates.
- **Source available.** We also specify whether the system's source code is available for download and is usable, that is, whether it can be compiled and run in modern platforms.

In the following we discuss further details of the state-of-the-art systems, grouped by their storage paradigms.

5.1.1. Independent Copies Systems

In an IC-like approach, each revision D_i of a dataset archive $\mathcal{A} = \{D_1, D_2, \ldots, D_n\}$ is fully stored as an independent RDF dataset. IC approaches shine at the execution of VM and CV queries as they do not incur any materialization cost for such types of queries. Conversely, IC systems are inefficient in terms of disk usage. For this reason they have mainly been proposed for small datasets or schema version control [34, 51]. SemVersion [51], for instance, is a system that offers similar functionalities as classical version control systems (e.g., CVS or SVN), with support for multiple RDF graphs and branching. In other words, SemVersion supports 5-tuples of the form $\langle s, p, o, g, i \rangle$, where g is a graph label and $i = rv(\rho)$. However, revision numbers are local to each RDF graph, which makes the system unable to track the addition or deletion of named graphs in the history of the dataset. Lastly, SemVersion provides an HTTP interface to submit updates either as RDF graphs or as changesets. Despite this flexibility, new revisions are always stored as independent copies. This makes its disk-space consumption prohibitive for large datasets like the ones studied in this paper.

5.1.2. Change-based Systems

Solutions based on the CB paradigm store a subset $\hat{\mathcal{A}} \subset \mathcal{A}$ of the revisions of a dataset archive as independent copies or *snapshots*. On the contrary, all the intermediate revisions D_j ($p < j < q$) between two snapshots D_p and D_q, are stored as deltas or changesets U_j. The sequence of deltas between two snapshots is called a *delta chain*. CB systems are convenient for DM and CD queries. Besides, they are obviously considerably more storage-efficient than IC solutions. Their weakness lies in the high materialization cost for VM and CV queries, particularly for long delta chains.

R&WBase [44] is a quad-based CB archiving solution that provides a Git-like interface with support for merging, branching, tagging, and concurrent updates on top of a classical SPARQL endpoint. R&WBase supports all types of archive queries on arbitrary BGPs. The system uses the PROV-Ontology (PROV-O) [12] to model the metadata (e.g., timestamps, parent branches) about the updates of a single RDF graph. An update u_i generates two new named graphs G_g^{i+}, G_g^{i-} containing the added and deleted triples in revision i. Revisions can be materialized by processing the delta chain back to the initial snapshot, and they can be referenced via aliases called *virtual named graphs*. In the same spirit, tags and branches are implemented as aliases of a particular revision. R&WBase has inspired the design of R43ples [24]. Unlike its predecessor, [24] can version multiple graphs, although revision numbers are not defined at the dataset level. Moreover, the system extends SPARQL with the clause *REVISION* j ($j \in \mathcal{N}$) used in conjunction with the GRAPH clause to match a BGP against a specific revision of a named graph. Last, the approach presented in [13] relies on an RDBMS to store snapshots and deltas of an RDF graph archive with support for branching and tagging. Its major drawback is the lack of support for SPARQL queries: while it supports all the types of queries intro-

	Storage paradigm	Data model	Full BGPs	Queries	Branch & tags	Multi-graph	Concurrent Updates	Source available
Dryda [2]	TB	5-tuples	+	all	-	+	-	-
Ostrich [49]	IC/CB/TB	quads	-	VM, DM, V	-	-	-	+
RDF-TX [22]	TB	quads	+	all	-	-	-	-
R43ples [24]	CB	5-tuples[c]	+	all	+	-	+	+[a]
R&WBase [44]	CB	quads	+	all	+	-	+	+[b]
RBDMS [13]	CB	quads	+	all	+	-	+	-
SemVersion [51]	IC	5-tuples[c]	-	VM, DM	+	-	+	-
v-RDFCSA [11]	TB	quads	-	VM, DM, V	-	-	-	-
x-RDF-3X [32]	TB	quads	+	VM, V	-	-	-	+[b]

[a] It needs modifications to have the console client running and working. [b] Old source code that does not work on modern systems.
[c] Graph local revisions

Table 2
Existing RDF archiving systems

duced in Section 2.5, they must be formulated in SQL, which can be very tedious for complex queries.

5.1.3. Timestamp-based Systems

TB solutions store triples with their temporal metadata, such as domain temporal validity intervals or insertion/deletion timestamps. Like in CB solutions, revisions must be materialized at a high cost for VM and CV queries. V queries are usually better supported, whereas the efficiency of materializing deltas depends on the system's indexing strategies.

x-RDF-3X [32] is a system based on the RDF-3X [31] engine. Logically x-RDF-3X supports quads of the form $\langle s, p, o, \rho \rangle$ where $\rho \in \mathcal{I}$ is associated to all the revisions where the triple was present as well as all its addition and deletion timestamps. The system is a fully-fledged query engine optimized for highly concurrent updates with support for snapshot isolation in transactions. However, x-RDF-3X does not support versioning for multiple graphs, neither branching nor tagging.

Dryda [2] is a TB archiving system that supports archiving of multi-graph datasets. Logically, Dryda stores 5-tuples of the form $\langle s, p, o, g, \zeta \rangle$, that is, it does not store local revision numbers for graphs. In its physical design, Dryda indexes quads $\langle s, p, o, g \rangle$ and associates them to visibility maps and creation/deletion timestamps, which determine the revisions and points in time where the quad was present. The system relies on six indexes – *gspo*, *gpos*, *gosp*, *spog*, *posg*, and *ospg* implemented as B+ trees – to support arbitrary SPARQL queries. Moreover, Dryda extends the query language with the clause *REVISION* x, where x can be a variable or a constant. This clause instructs the query engine to match a BGP against the contents of the data sources bound to x, namely a single revision ζ, or a delta $U_{j,k}$. A revision can be identified by its IRI ζ, its revision number $j = rv(\zeta)$ or by a timestamp τ'. The latter case matches the revision ζ with the largest timestamp $\tau = ts(\zeta)$ such that $\tau \leqslant \tau'$. Alas, Dryda's source is not available for download and use.

RDF-TX [22] supports single RDF graphs and uses a multiversion B-tree (MVBT) to index triples and their time metadata (insertion and deletion timestamps). An MVBT is actually a forest where each tree stores the triples that were inserted within a time interval. RDF-TX implements an efficient compression scheme for MVBTs, and proposes SPARQL-T, a SPARQL extension that adds a fourth component $\hat{g}$ to BGPs. This component can match only time objects τ of type timestamp or time interval. The attributes of such objects can be queried via built-in functions, e.g., $year(\tau)$. While RDF-TX offers interval semantics at the query level, it stores only timestamps.

v-RDFCSA [11] is a lightweight and storage-efficient TB approach that relies on suffix-array encoding [7] for efficient storage with basic retrieval capabilities (much in the spirit of HDT [16]). Each triple is associated to a bitsequence of length equals the number of revisions in the archive. That is, v-RDFCSA logically stores quads of the form $\langle s, p, o, rv(\rho) \rangle$. Its query functionalities are limited since it supports only VM, DM, and V queries on single triple patterns.

5.1.4. Hybrid Systems

Some approaches can combine the strengths of the different storage paradigms. One example is Ostrich [49], which borrows inspirations from IC, CB, and TB systems. Logically, Ostrich supports quads of the form $\langle s, p, o, rv(\rho) \rangle$. Physically, it stores snapshots of

an RDF graph using HDT [16] as serialization format. Delta chains are stored as B+ trees time-stamped with revision numbers in a TB-fashion. These delta chains are redundant, i.e., each delta stores the changes w.r.t. the latest snapshot (and not the previous revision as also proposed in [13]). Ostrich alleviates the cost of redundancy using compression. All these design features make Ostrich query and space efficient, however its functionalities are limited. Its current implementation does not support more than one (initial) snapshot. Multi-graph archiving as well as branching/tagging are not possible. Moreover, the system's querying capabilities are restricted to VM, DM, and V queries on single triple patterns.

5.2. Languages to Query RDF Archives

Multiple research endeavors have proposed alternatives to succinctly formulate queries on RDF archives. [18] uses AnQL to express the query types described in Section 2.5. AnQL [52] is a SPARQL extension based on quad patterns $\langle \hat{s}, \hat{p}, \hat{o}, \hat{g} \rangle$. AnQL is more general than SPARQL-T (proposed by RDF-TX [22]) because the $\hat{g}$-component can be bound to any term $u \in \mathcal{I} \cup \mathcal{L}$ (not only time objects). For instance, a DM query asking for the countries added at revision 1 to our example RDF dataset from Figure 1 could be written as follows:

```
SELECT * WHERE {
 { (?x a :Country): [1] } MINUS
 { (?x a :Country) :[0] }
}
```

T-SPARQL [23] is a SPARQL extension inspired by the query language TSQL2 [46]. T-SPARQL allows for the annotation of groups of triple patterns with constraints on temporal validity and commit time, i.e., it supports both time-intervals and timestamps as time objects. T-SPARQL defines several comparison operators between time objects, namely *equality*, *precedes*, *overlaps*, *meets*, and *contains*. Similar extensions [4, 39] also offer support for geo-spatial data.

SPARQ-LTL [19] is a SPARQL extension that makes two assumptions, namely that (i) triples are annotated with revision numbers, and (ii) revisions are accessible as named graphs. When no named graph is specified, BGPs are iteratively matched against *every* revision. A set of clauses on BGPs can instruct the SPARQL engine to match a BGP against other revisions at each iteration. For instance the clause *PAST* in the expression *PAST*{ q } MINUS { q } with $q = \langle ?x$ *a :Country*$\rangle$ will bind variable ?x to all the countries that were ever deleted from the RDF dataset, even if they were later added.

5.3. Benchmarks and Tools for RDF Archives

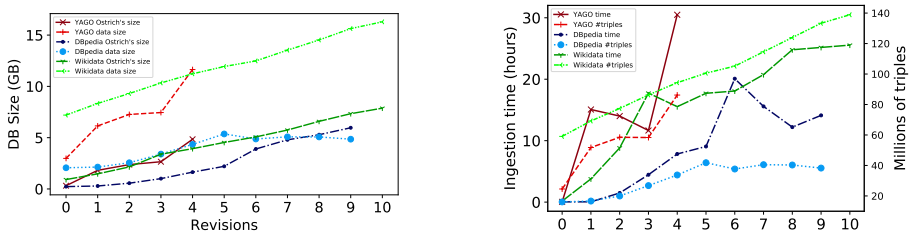
BEAR [18] is the state-of-the-art benchmark for RDF archive solutions. The benchmark provides three real-world RDF graphs (called BEAR-A, BEAR-B, and BEAR-C) with their corresponding history, as well as a set of VM, DM, and V queries on those histories. In addition, BEAR allows system designers to compare their solutions with baseline systems based on different storage strategies (IC, CB, TB, and hybrids TB/CB, IC/CB) and platforms (Jena TDB and HDT). Despite its multiple functionalities and its dominant position in the domain, BEAR has some limitations: (i) It assumes single-graph RDF datasets; (ii) it does not support CV and CD queries, moreover VM, DM, and V queries are defined on single triple patterns; and (iii) it cannot simulate datasets of arbitrary size and query workloads.

EvoGen [29] tackles the latter limitation by extending the Lehigh University Benchmark (LUBM) [25] to a setting where both the schema and the data evolve. Users can not only control the size and frequency of that evolution, but can also define customized query workloads. EvoGen supports all the types of queries on archives presented in Section 2.5 on multiple triple patterns.

A recent approach [50] proposes to use FCA (Formal Concept Analysis) and several data fusion techniques to produce summaries of the evolution of entities across different revisions of an RDF archive. A summary can, for instance, describe groups of subjects with common properties that change over time. Such summaries are of great interest for data maintainers as they convey edition patterns in RDF data through time.

6. Related Work Evaluation

In this section we conduct an evaluation of the state-of-the-art RDF archiving engines. We first provide a global analysis of the systems' functionalities in Section 6.1. Section 6.2 then provides a performance evaluation of Ostrich (the only testable solution) on our experimental RDF archives from Table 1. This evaluation is complementary to the Ostrich's evaluation on BEAR (available at [49]), as it shows the performance of the system in three real-world large RDF datasets.



(a) Space used by Ostrich to stores datasets (b) Time used by Ostrich to ingest each versions

Fig. 6. Ostrich's performance with DBpedia and YAGO

6.1. *Functionality Analysis*

As depicted in Table 2, existing RDF archiving solutions differ greatly in design and functionality. The first works [10, 32, 51] offered mostly storage of old revisions and support for basic VM queries. Consequently, subsequent efforts focused on extending the query capabilities and allowing for concurrent updates as in standard version control systems [13, 24, 44]. Such solutions are attractive for data maintainers in collaborative projects, however they still lack scalability, e.g., they cannot handle large datasets and changesets, besides conflict management is still delegated to users. More recent works [11, 49] have therefore focused on improving storage and querying performance, alas, at the expense of features. For example, Ostrich [49] does not support queries on arbitrary BGPs and is limited to a single snapshot. In addition to the limitations in functionality, Table 2 shows that most of the existing systems are not easily available, either because their source code is not directly usable, or because they do not easily compile in modern platforms. While this still leaves us with Ostrich [49] and R43ples [24] as testable solutions, only [49] was able to run on our experimental datasets [6].

6.2. *Performance Analysis*

We evaluate the performance of Ostrich on our experimental datasets in terms of storage space and ingestion time, i.e., the time to generate a new revision from an input changeset, and query response time. Figure 6a shows the amount of storage space (in GB) used by Ostrich for the selected revisions of our experimental datasets. We provide the raw sizes of the RDF dumps of each revision for reference. Storing each version of YAGO separately requires 36 GB, while Ostrich uses only 4.84 GB. For DBpedia compression goes from 39 GB to 5.96 GB. As for Wikidata, its take 131 GB to stores the raw files, but only 7.88 GB with Ostrich. This yields a compression rate of 87% for YAGO, 84% for DBpedia and 94% for Wikidata. This space efficiency is the result of using HDT [16] for snapshot storage, as well as compression for the delta chains.

Figure 6b shows Ostrich's ingestion times. We also provide the number of triples of each revision as reference. The results suggest that this measure depends both on the changeset size, and the length of the delta chain. However, the latter factor becomes more prominent as the length of the delta chain increases. For example, we can observe that Ostrich requires ∼22 hours to ingest revision 9 of DBpedia (2.43M added and 2.46M deleted triples) while it takes only ∼14 hours to ingest revision 5 (12.85M added and 5.95M deleted triples). This confirms the trends observed in [49] where ingestion time increases linearly with the number of revisions. Overall, Ostrich proves to handle large deltas in a reasonable amount of time, i.e., no more than 30 hours for large changesets of up to 51M triples (YAGO).

Table 3 shows Ostrich's average runtime in seconds for VM, V, and DM queries on our experimental datasets. We group the runtimes by type of single triple pattern (as Ostrich does not support arbitrary BGPs). Each cell in Table 3 corresponds to the average across 100 queries, where both entities in triple patterns and revision numbers were randomly generated. We set a timeout of 1 hour for each query, and show the number of timeouts in parentheses next to the runtime – which excludes queries that timed out. The category labeled *top p* consists of queries on the top five most frequent predicates in each dataset. We observe that Ostrich is roughly one order of magnitude faster on YAGO than on DBpedia and Wikidata. This can be the effect of

[6] R43ples is unable to ingest large files as it can lead to the creation of arrays exceeding the maximum array size of Java

Triple Patterns	DBpedia VM	DBpedia V	DBpedia DM	YAGO VM	YAGO V	YAGO DM	Wikidata VM	Wikidata V	Wikidata DM
? p ?	92.81(0)	118.64(0)	91.78(0)	2.9(3)	- (5)	1.82(3)	281.41(0)	302.26(0)	303.73(0)
? <top p> o	112.81(0)	283.59(0)	130.88(0)	35.08(2)	69.65(4)	137.16(4)	347.06(0)	499.77(0)	285.02(0)
? p o	96.74(0)	92.04(0)	91.93(0)	2.38(4)	2.35(4)	2.35(2)	282.12(0)	281.63(0)	281.45(0)
s p ?	94.99(0)	91.67(0)	92.81(0)	2.42(2)	2.36(3)	2.41(1)	284.74(0)	281.4(0)	281.2(0)

Table 3
Ostrich's Query Performance in seconds

the longer delta chains of the two latter datasets. Despite the overall faster runtime on YAGO, we observe some timeouts in contrast to DBpedia and Wikidata. We believe this is caused by the sizes of the changesets, which are of 3.72GB on YAGO, versus 2.09GB in DBpedia and 1.86GB in Wikidata. YAGO's changesets at revisions 1 and 4 are very large as shown in Section 4.

All the experiments were run on a server with a 4-core CPU (Intel Xeon E5-2680 v3@2.50GHz) and 64 GB of RAM.

7. Towards Fully-fledged RDF Archiving

In this section we use the insights from previous sections to derive a set of lessons towards the design of a fully-fledged solution for archiving of large RDF datasets.

Global and local history. Our survey in Section 5.1 suggests that none of the available solutions supports both archiving of the local and joint (global) history of multiple RDF graphs. We argue that such a feature would be of great value for users and maintainers of data warehouses.

Temporal domain-specific vs. revision metadata. Solutions and language extensions for temporal domain-specific queries [22, 23] that go beyond the , treat both domain-specific metadata (e.g., triple validity intervals) and revision-related annotations (e.g., revision numbers) in the same way. We highlight, however, that revision metadata is immutable and should therefore be logically placed at a different level. In this line of thought we propose to associate revision metadata for graphs and datasets, e.g., commit time, revision numbers, or branching & tagging information, to the local and global revision identifiers ρ and ζ, whereas domain-specific time objects should be modeled as statements about the graph labels $g = l(\rho)$.

Storage paradigm. RDF archiving differs from standard RDF management in an even more imperative need for storage efficiency. As shown in [49], the existing storage paradigms shine at different types of queries. Hence, supporting arbitrary queries while being storage-efficient requires the best from the IC, CB, and TB philosophies. A hybrid approach, however, will inevitably be more complex and will introduce further parameters and trade-offs. For example, CB solutions must define a policy to decide when to store a revision as a snapshot or as a delta, trading disk space for lower response time for VM queries. Likewise, TB solutions must build further indexes to efficiently answer V queries.

Accounting for evolution patterns. As our study in Section 4 shows, the evolution patterns of RDF archives can change throughout time. With that in mind, we envision an adaptive data-oriented system that relies on the metrics proposed in Section 3 and adjusts its parameters according to the archive's evolution. For example, a decreasing trend in vocabularity dynamicity could automatically trigger a more conservative allocation of the storage resources for single entity indexes. In the same spirit, large changesets could translate into shorter delta chains so that VM queries are executed efficiently. Moreover, such an adaptive approach could be implemented in a query-aware fashion whenever the query load is known. Finally as we expect long dataset histories, it is vital for solutions to improve their ingestion time complexity, which should depend more on the size of the changesets rather than in the history size, contrary to what we observed in Section 6.

Concurrent updates. We can group the existing state-of-the-art solutions in three categories regarding their support for concurrent updates, namely (i) solutions with limited or no support for concurrent updates [2, 11, 22, 40, 49], (ii) solutions inspired by version control systems such as Git [13, 24, 44, 51], and (iii) approaches with full support for highly concurrent updates [32]. Git-like solutions are particularly interest-

ing for collaborative efforts such as DBpedia, because it is feasible to delegate users the task of conflict management. Conversely, fully automatically constructed KBs such as [9] or data-intensive (e.g., streaming) applications may need the features of solutions such as [32]. Consequently, we propose to separate the concurrency layer from the storage backend. Such a middleware could take care of enforcing a consistency model for concurrent commits either automatically or via user-based conflict management. The layer could also manage the additional metadata for features such as branching and tagging.

Serialization and querying. Archiving of multi-graph datasets requires a solution for serialization and querying for two levels of metadata for triples. Existing solutions include reification [42], singleton properties [33], and named graphs [41]. Reification assigns each RDF statement (triple or quad) an identifier $t \in \mathcal{I}$ that can be then used to link the triple to its ρ and ζ components in the 5-tuples data model introduced in Section 2.3. While simple and fully compatible with the existing RDF standards, reification is well-known to incur serious performance issues for storage and query efficiency, e.g., it would quintuple the number of triple patterns in SPARQL queries. On those grounds [33] proposes singleton properties to piggyback the metadata in the predicate component. In this vibe, predicates take the form $p\#m \in \mathcal{I}$ for some $m \in \mathcal{N}$ and every triple with p in the dataset. This scheme gives $p\#m$ the role of ρ in our data model reducing the overhead of reification. However, singleton properties would still require an additional level of reification for the fifth component ζ. The same is true for a solution based on named graphs. For all these reasons, we believe that RDF* [26] is an attractive solution for fully-fledged RDF archiving. This extension of the RDF data model proposes nested triples, that is, the subject and object of a triple can also be triples. For example RDF* could serialize the tuple $\langle$*:Peter*, *:spouse*, *:Sasha*, $\rho, \zeta\rangle$ with local (*:lrv*) and global (*:grv*) revision numbers $rv(\rho)$=1, $rv(\zeta) = 2$ as follows:

<<<:Peter :spouseOf :Sasha> :lrv 1> :grv 2>

The authors of [26] propose this serialization as part of the Turtle* format. Moreover, they propose SPARQL* that allows for nested triple patterns. While SPARQL* enables the definition of metadata constraints at different levels, a fully archive-compliant language could offer further syntactic sugar such as the clauses *RE-VISION* (proposed by [2, 24]) or *DELTA* to bind the variables of a BGP to the data in particular revisions or deltas. We propose to build such an archive-compliant language upon SPARQL*.

8. Conclusions

In this paper we have argued the importance of RDF archiving for both maintainers and consumers of RDF data. Besides, we have argued the importance of evolution patterns in the design of a full-fledged RDF archiving solution. On those grounds, we have proposed a metric-based framework to characterize the evolution of RDF data, and we have applied our framework to study the history of three challenging RDF datasets, namely DBpedia, YAGO, and Wikidata. Our survey and study of existing solutions and benchmarks for RDF archiving has shown that they are still far from tackling the major challenges of this task, mainly because, (i) they do not account for evolution patterns of RDF data, (ii) they do not provide native support for arbitrary SPARQL queries on archives, and (iii) they struggle with the scale of current real-world datasets. We have used all these observations to derive a set of design lessons in order to overcome the gap between the literature and a fully functional solution for large RDF archives. With this detailed study, we aim at paving the way towards a ultimate solution for this problem.

Acknowledgements

This research was partially funded by the Danish Council for Independent Research (DFF) under grant agreement no. DFF-8048-00051B and the Poul Due Jensen Foundation.

References

[1] RDF Exports from Wikidata. Available at tools.wmflabs.org/wikidata-exports/rdf/index.html.

[2] James Anderson and Arto Bendiken. Transaction-time queries in dydra. In *MEPDaW/LDQ@ESWC*, volume 1585 of *CEUR Workshop Proceedings*, pages 11–19. CEUR-WS.org, 2016.

[3] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A Nucleus for a Web of Open Data. In *The Semantic Web*, pages 722–735. Springer, 2007.

[4] Konstantina Bereta, Panayiotis Smeros, and Manolis Koubarakis. Representation and querying of valid time of triples in linked geospatial data. In *Extended Semantic Web Conference*, pages 259–274. Springer, 2013.

[5] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The Semantic Web. *Scientific American*, 284(5):28–37, 2001.

[6] Christian Bizer. The Emerging Web of Linked Data. *IEEE Intelligent Systems*, 24(5):87–92, 2009.

[7] Nieves R. Brisaboa, Ana Cerdeira-Pena, Antonio Fariña, and Gonzalo Navarro. A compact rdf store using suffix arrays. In Costas Iliopoulos, Simon Puglisi, and Emine Yilmaz, editors, *String Processing and Information Retrieval*, pages 103–115, Cham, 2015. Springer International Publishing.

[8] JÃűrg Brunsmann. Archiving pushed inferences from sensor data streams. In *Proceedings of the International Workshop on Semantic Sensor Web - Volume 1: SSW, (IC3K 2010)*, pages 38–46. INSTICC, SciTePress, 2010.

[9] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka Jr., and Tom M. Mitchell. Toward an architecture for never-ending language learning. In *Proceedings of the Twenty-Fourth Conference on Artificial Intelligence*, AAAI, 2010.

[10] Steve Cassidy and James Ballantine. Version control for rdf triple stores. *ICSOFT (ISDM/EHST/DC)*, 7:5–12, 2007.

[11] Ana Cerdeira-Pena, Antonio Fariña, Javier D. Fernández, and Miguel A. Martínez-Prieto. Self-indexing RDF archives. In *DCC*, pages 526–535. IEEE, 2016.

[12] The World Wide Web Consortium. Prov-o: The prov ontology. www.w3.org/TR/prov-o, 2013.

[13] Dong-hyukim, Sang-wonlee, and Hyoung-jookim. A version management framework for RDF triple stores. *International Journal of Software Engineering and Knowledge Engineering*, 22, 04 2012.

[14] Ivan Ermilov, Jens Lehmann, Michael Martin, and Sören Auer. LODStats: The Data Web Census Dataset. In *Proceedings of 15th International Semantic Web Conference - Resources Track (ISWC'2016)*, 2016.

[15] Fredo Erxleben, Michael Günther, Markus Krötzsch, Julian Mendez, and Denny Vrandečić. Introducing Wikidata to the Linked Data Web. In *International Semantic Web Conference*, pages 50–65. Springer, 2014.

[16] Javier D. Fernández, Miguel A. Martínez-Prieto, Claudio Gutiérrez, Axel Polleres, and Mario Arias. Binary rdf representation for publication and exchange (hdt). *Web Semantics: Science, Services and Agents on the World Wide Web*, 19:22–41, 2013.

[17] Javier D Fernández, Axel Polleres, and Jürgen Umbrich. Towards efficient archiving of dynamic linked open data. *Proceedings of the First DIACHRON Workshop on Managing the Evolution and Preservation of the Data Web*, pages 34–49, 2015.

[18] Javier D Fernández, Jürgen Umbrich, Axel Polleres, and Magnus Knuth. Evaluating query and storage strategies for rdf archives. In *Proceedings of the 12th International Conference on Semantic Systems*, pages 41–48. ACM, 2016.

[19] Valeria Fionda, Melisachew Wudage Chekol, and Giuseppe Pirrò. Gize: A time warp in the web of data. In *International Semantic Web Conference (Posters & Demos)*, volume 1690 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2016.

[20] The Apache Software Foundation. Apache jena semantic web framework. jena.apache.org.

[21] Luis Galárraga, Kim Ahlstrøm, Katja Hose, and Torben Bach Pedersen. Answering provenance-aware queries on rdf data cubes under memory budgets. In *The Semantic Web – ISWC 2018*, pages 547–565, Cham, 2018. Springer International Publishing.

[22] Shi Gao, Jiaqi Gu, and Carlo Zaniolo. Rdf-tx: A fast, user-friendly system for querying the history of rdf knowledge bases. In *EDBT*, pages 269–280, 2016.

[23] Fabio Grandi. T-SPARQL: A tsql2-like temporal query language for RDF. In *Local Proceedings of the Fourteenth East-European Conference on Advances in Databases and Information Systems*, pages 21–30, 2010.

[24] Markus Graube, Stephan Hensel, and Leon Urbas. R43ples: Revisions for triples. In *Proceedings of the 1st Workshop on Linked Data Quality co-located with 10th International Conference on Semantic Systems (SEMANTiCS 2014)*, 2014.

[25] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. Lubm: A benchmark for owl knowledge base systems. *Journal of Web Semantics*, 3(2):158–182, 2005. Selcted Papers from the International Semantic Web Conference, 2004.

[26] Olaf Hartig. Foundations of RDF⋆ and SPARQL⋆: An Alternative Approach to Statement-Level Metadata in RDF. In *Proc. of the 11th Alberto Mendelzon International Workshop on Foundations of Data Management*, AMW, 2017.

[27] Ian Horrocks, Thomas Hubauer, Ernesto Jiménez-Ruiz, Evgeny Kharlamov, Manolis Koubarakis, and Ralf M˙ Addressing Streaming and Historical Data in OBDA Systems: Optique's Approach (Statement of Interest).

[28] Thomas Huet, Joanna Biega, and Fabian M. Suchanek. Mining history with le monde. In *Proceedings of the 2013 Workshop on Automated Knowledge Base Construction*, AKBC '13, pages 49–54, New York, NY, USA, 2013. ACM.

[29] Marios Meimaris. Evogen: a generator for synthetic versioned RDF. In Themis Palpanas and Kostas Stefanidis, editors, *Proceedings of the Workshops of the EDBT/ICDT 2016 Joint Conference, EDBT/ICDT Workshops 2016, Bordeaux, France, March 15, 2016*, volume 1558 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2016.

[30] Rudra Pratap Deb Nath, Katja Hose, Torben Bach Pedersen, Oscar Romero, and Amrit Bhattacharjee. Set$_{LBI}$: An Integrated Platform for Semantic Business Intelligence. In *Proceedings of The 2020 World Wide Web Conference*.

[31] Thomas Neumann and Gerhard Weikum. Rdf-3x: a risc-style engine for rdf. *Proceedings of the VLDB Endowment*, 1(1):647–659, 2008.

[32] Thomas Neumann and Gerhard Weikum. x-rdf-3x: fast querying, high update rates, and consistency for rdf databases. *Proceedings of the VLDB Endowment*, 3(1-2):256–263, 2010.

[33] Vinh Nguyen, Olivier Bodenreider, and Amit Sheth. Don't Like RDF Reification?: Making Statements About Statements Using Singleton Property. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, pages 759–770, New York, NY, USA, 2014. ACM.

[34] Natasha Noy and Mark Musen. Ontology versioning in an ontology management framework. *Intelligent Systems, IEEE*, 19:6 – 13, 08 2004.

[35] George Papadakis, Konstantina Bereta, Themis Palpanas, and Manolis Koubarakis. Multi-core meta-blocking for big linked data. In *Proceedings of the 13th International Conference on Semantic Systems*, Semantics2017, pages 33–40, New York, NY, USA, 2017. ACM.

[36] Vicky Papavassiliou, Giorgos Flouris, Irini Fundulaki, Dimitris Kotzinos, and Vassilis Christophides. On detecting high-level changes in rdf/s kbs. In *International Semantic Web Conference (ISWC)*, pages 473–488, 2009.

[37] Thomas Pellissier Tanon, Camille Bourgaux, and Fabian Suchanek. Learning how to correct a knowledge base from the edit history. In *The World Wide Web Conference*, WWW '19, pages 1465–1475, New York, NY, USA, 2019. ACM.

[38] Thomas Pellissier Tanon and Fabian M. Suchanek. Querying the Edit History of Wikidata. In *Extended Semantic Web Conference*, Portorož, Slovenia, 2019.

[39] Matthew Perry, Prateek Jain, and Amit P. Sheth. SPARQL-ST: Extending SPARQL to Support Spatio-temporal Queries. In *Geospatial Semantics and the Semantic Web*, volume 12 of *Semantic Web and Beyond: Computing for Human Experience*, pages 61–86. Springer, 2011.

[40] Maria Psaraki and Yannis Tzitzikas. Cpoi: A compact method to archive versioned rdf triple-sets, 2019.

[41] Yves Raimond and Guus Schreiber. RDF 1.1 primer. W3C recommendation, 2014. http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/.

[42] Yves Raimond and Guus Schreiber. RDF 1.1 semantics. W3C recommendation, 2014. http://www.w3.org/TR/rdf11-mt/.

[43] Yannis Roussakis, Ioannis Chrysakis, Kostas Stefanidis, Giorgos Flouris, and Yannis Stavrakas. A flexible framework for understanding the dynamics of evolving rdf datasets. In *International Semantic Web Conference (ISWC)*, 2015.

[44] Miel Vander Sande, Pieter Colpaert, Ruben Verborgh, Sam Coppens, Erik Mannens, and Rik Van de Walle. R&wbase: git for triples. *Linked Data on the Web Workshop*, 2013.

[45] Andy Seaborne and Steven Harris. SPARQL 1.1 query language. W3C recommendation, W3C, 2013. http://www.w3.org/TR/2013/REC-sparql11-query-20130321/.

[46] Richard T. Snodgrass, Ilsoo Ahn, Gad Ariav, Don S. Batory, James Clifford, Curtis E. Dyreson, Ramez Elmasri, Fabio Grandi, Christian S. Jensen, Wolfgang Käfer, Nick Kline, Krishna G. Kulkarni, T. Y. Cliff Leung, Nikos A. Lorentzos, John F. Roddick, Arie Segev, Michael D. Soo, and Suryanarayana M. Sripada. TSQL2 language specification. *SIGMOD Record*, 23(1):65–86, 1994.

[47] OpenLink Software. Openlink virtuoso. virtuoso.openlinksw.com.

[48] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: A Large Ontology from Wikipedia and Wordnet. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(3):203–217, 2008.

[49] Ruben Taelman, Miel Vander Sande, and Ruben Verborgh. OSTRICH: versioned random-access triple store. In *Companion of the The Web Conference 2018, WWW 2018, Lyon , France, April 23-27, 2018*, pages 127–130, 2018.

[50] Mayesha Tasnim, Diego Collarana, Damien Graux, Fabrizio Orlandi, and Maria-Esther Vidal. Summarizing entity temporal evolution in knowledge graphs. In *Companion Proceedings of The 2019 World Wide Web Conference*, WWW âĂŹ19, page 961âĂŞ965, New York, NY, USA, 2019. Association for Computing Machinery.

[51] Max Volkel, Wolf Winkler, York Sure, Sebastian Ryszard Kruk, and Marcin Synak. SemVersion: A Versioning System for RDF and Ontologies. *Second European Semantic Web Conference*, 2005.

[52] Antoine Zimmermann, Nuno Lopes, Axel Polleres, and Umberto Straccia. A general framework for representing, reasoning and querying with annotated semantic web data. *Web Semant.*, 11:72–95, March 2012.