Highlights

Performing live time-traversal queries via SPARQL on RDF datasets

Arcangelo Massari, Silvio Peroni

- The most prominent knowledge bases do not support SPARQL time-traversal queries on previous statuses of their entities together with provenance information
- The OpenCitations provenance model can manage both provenance and change-tracking, complying with RDF 1.1
- time-agnostic-library enables all the time-related retrieval functionalities via SPARQL live

Performing live time-traversal queries via SPARQL on RDF datasets

Arcangelo Massari[a,*], Silvio Peroni[a,b]

[a]*Research Centre for Open Scholarly Metadata, Department of Classical Philology and Italian Studies, University of Bologna, Via Zamboni, 33, Bologna, 40126, Italy*
[b]*Digital Humanities Advanced Research Centre (/DH.arc), Department of Classical Philology and Italian Studies, Via Zamboni, 33, Bologna, 40126, Italy*

Abstract

This article introduces a methodology to perform live time-traversal SPARQL queries on RDF datasets and software based on this methodology that offers a solution to manage the provenance and change-tracking of entities described using RDF. These are crucial factors in ensuring verifiability and trust. Nevertheless, some of the most prominent knowledge bases – including DBpedia, Wikidata, Yago, and the Dynamic Linked Data Observatory – do not support time-agnostic queries, i.e., queries across different snapshots together with provenance information. The OpenCitations Data Model (OCDM) describes one possible way to track provenance and entities' changes in RDF datasets, and it allows restoring an entity to a specific status in time (i.e., a snapshot) by applying SPARQL update queries. The methodology and library presented in this article are based on the rationale introduced in the OCDM. We also developed benchmarks proving that such a procedure is efficient for specific queries and less efficient for others. To the best of our knowledge, our library is the only one to support all the time-related retrieval functionalities live, i.e., enabling real-time searches and updates. Moreover, since OCDM complies with standard RDF, queries are expressed via standard SPARQL.

Keywords: provenance, change-tracking, time-traversal queries, RDF, SPARQL, OpenCitations

1. Introduction

Data reliability is based on provenance: who produced information, when, and the primary source. Such provenance information is essential because the truth value of an assertion on the Web is never absolute, as claimed by Wikipedia, which on its policy on the subject states: "the threshold for inclusion in Wikipedia is verifiability, not truth" [1]. The Semantic Web reinforces this aspect since each application processing information must evaluate trustworthiness by probing the statements' context (i.e., the provenance) [2].

Moreover, data changes over time, for either the natural evolution of concepts or the correction of mistakes. Indeed, the latest version of knowledge may not be the most accurate. Such phenomena are particularly tangible in the Web of Data, as highlighted in a study by the Dynamic Linked Data Observatory, which noted the modification of about 38% of the nearly 90,000 RDF documents monitored for 29 weeks and the permanent disappearance of 5% [3].

Notwithstanding these premises, the most extensive RDF datasets to date – DBPedia, Wikidata, Yago, and the Dynamic Linked Data Observatory – either do not use RDF to track changes or do not provide provenance information at the entity level [4–7]. Therefore, they don't allow SPARQL time-traversal queries on previous statuses of their entities together with provenance information. For

*Corresponding author
Email addresses: arcangelo.massari@unibo.it (Arcangelo Massari), silvio.peroni@unibo.it (Silvio Peroni)

instance, Wikidata allows SPARQL queries on entities temporally annotated via its proprietary RDF extension but does not allow queries on change-tracking data.

The main reason behind this phenomenon is that the founding technologies of the Semantic Web – SPARQL, OWL, and RDF – did not initially provide an effective mechanism to annotate statements with metadata information. This lacking led to the introduction of numerous metadata representation models, none of which succeeded in establishing itself over the others and becoming a widely accepted standard to track both provenance and changes to RDF entities [8–26].

In the past, some software was developed to perform time-traversal queries on RDF datasets, enabling the reconstruction of the status of a particular entity at a given time. However, some existing solutions need to preprocess and index RDF data to work efficiently [27–31]. This requirement is impractical for linked open datasets that constantly receive many updates, such as Wikidata. For example, "Ostrich requires $\sim$ 22 hours to ingest revision 9 of DBpedia (2.43M added and 2.46M deleted triples)" [32]. Conversely, software operating on the fly either does not support all query types [33], or supports them non-generically by imposing a custom database [34] or a specific triplestore [35, 36].

This work introduces a methodology and a Python library enabling all the time-related retrieval functionalities identified by Fernández et al. [37] live, i.e., allowing real-time queries and updates without preprocessing the data. Moreover, data can be stored on any RDF-compliant storage system (e.g., RDF-serialized textual files and triplestores) when the provenance and data changes are tracked according to the OpenCitations Data Model [38].

The rest of the paper is organized as follows. Section 2 reviews the literature on metadata representation models, retrieval functionalities, and archiving policies for dynamic linked data. Section 3 showcases the methodology underlying the *time-agnostic-library* implementation, and Section 4 discusses the final product from a quantitative point of view, reporting the benchmarks results on execution times and memory.

2. Related works

This section reviews related metadata representation models (Section 2.1) before delving into query typologies, query languages (Section 2.2), and existing methodologies to performing such queries (Section 2.3).

2.1. Representing dynamic linked data

The landscape of strategies to formally represent provenance in RDF is vast and fragmented [39].

To date, the only W3C standard syntax for annotating triples' provenance is RDF reification [40] and it is the only one to be back-compatible with all RDF-based systems. However, there are several deprecation proposals for this syntax [41], due to its poor scalability.

Different approaches have been proposed since 2005, and four categories of solutions can be identified:

- Encapsulating provenance in RDF triples: n-ary relations [42], PaCE [16] and singleton properties [17].
- Associating provenance to the triple through RDF quadruples: named graphs [8], RDF/S graphsets [9], RDF triple coloring [10], and nanopublications [43].
- Extending the RDF data model: Notation 3 Logic [11], RDF+ [12], SPOTL(X) [14], annotated RDF (aRDF) [13, 44], and RDF* [15].
- Using ontologies: Proof Markup Language [21], SWAN Ontology [25], Provenir Ontology [23], Provenance Ontology [45], Open Provenance Model [20], PREMIS [24], Dublin Core Metadata Terms [26], and the OpenCitations Data Model [38].

For a complete analysis and comparison, refer to Sikos & Philp [39]. In this context it is important to stress that

most of these solutions do not comply with RDF 1.1 (i.e., RDF/S graphsets, N3Logic, aRDF, RDF+, SPOTL(X), and RDF*), are domain-specific (i.e., Provenir, SWAN, and PREMIS ontologies), rely on blank nodes (n-ary relations), or suffer from scalability issues (singleton properties, PaCE).

Despite being incompatible with RDF 1.1, it is worth mentioning that a W3C working group has recently published the first draft to make RDF* a standard [46].

To date, named graphs [8] and the Provenance Ontology [47] are the most adopted approaches to attach provenance metadata to RDF triples. On the one hand, Named Graphs are widespread because they are compliant with RDF 1.1 and can be queried with SPARQL 1.1; they are scalable, and have several serialization formats (i.e., TriX, TriG, and N-Quads). On the other, the Provenance Ontology was published by the Provenance Working Group as a W3C Recommendation in 2013, meeting all the requirements for provenance on the Web and collecting existing ontologies into a single general model.

The OpenCitations Data Model [38] represent provenance and track changes in a way that complies with RDF 1.1 and relies on well-known and widely adopted standards, PROV-O, named graphs, and Dublin Core, as will be detailed in Section 3.

2.2. Querying dynamic linked data

Fernández, Polleres, and Umbrich [37] provided two classifications on time agnostic queries, a low-level one relating to "query atoms" and a high-level one about "retrieval needs". In this article, we use the high-level classification, which is more explicit about the queries to reconstruct a full version of an entity, an entire delta, and the query on multiples/all deltas, without the need to derive them by composition between multiple queries atoms. Before detailing such queries, it is required to define what an entity, a time-aware dataset, and a version are.

Definition 1 (Entity). An entity $\mathcal{E}$ is the set of RDF triples (s,p,o) having the same subject s.

Definition 2 (Time-aware dataset). A version annotated entity is an entity $\mathcal{E}$ annotated with a label i representing the version in which this entity holds, denoted by the notation $\mathcal{E}_i$, where $i \in \mathcal{N}$. A time-aware dataset $\mathcal{A}$ is a set of version-annotated entities.

Definition 3 (Version). A version of a time-aware dataset $\mathcal{A}$ at snapshot i is the RDF graph $\mathcal{A}_i = \{\mathcal{E}|\mathcal{E}_i \in \mathcal{A}\}$.

In the query definitions, the evaluation of a SPARQL query Q on a graph $\mathcal{G}$ produces a bag of solution mappings $[[Q]]_{\mathcal{G}}$.

Version materialization (VM) retrieves the full version of a specific entity. Formally: $VM(\mathcal{E}, i) = \mathcal{E}_i$. For example, "Get the 2014 snapshot of the entity representing David Shotton".

Single-version structured query (SV) retrieves the results of a SPARQL query targeted at a specific version. Formally: $SV(Q, V_i) = [[Q]]_{V_i}$. For example, "Which David Shotton's papers were featured in the dataset in 2014?".

Cross-version structured query (CV) — also called time-traversal query — retrieves the results of a SPARQL query targeted at multiple versions. Formally: $CV(Q, V_i, V_j) = SV(Q, V_i) \bowtie SV(Q, V_j)$. For example, "Which David Shotton's papers were featured in the dataset in 2013 and in 2014?".

Delta materialization (DM) retrieves the differences of a specific entity between two consecutive versions. Formally: $DM(\mathcal{E}, V_i) = (\Delta+, \Delta-)$. With $\Delta+ = \mathcal{E}_i \backslash \mathcal{E}_j$, $\Delta- = \mathcal{E}_j \backslash \mathcal{E}_i$ and $i, j \in \mathcal{N}, i > j, \nexists k \in \mathcal{N} : j < k < i$. For example, "What data changed about the entity representing David Shotton in 2014?".

Single-delta structured query (SD) retrieves the change-sets of a SPARQL query's results between one consecutive couple of versions. Formally: $SD(Q, V_i, V_j) = (\Delta+, \Delta-)$. With $\Delta+ = [[Q]]_{V_i} \backslash [[Q]]_{V_j}$, $\Delta- = [[Q]]_{V_j} \backslash [[Q]]_{V_i}$

and $i, j \in \mathcal{N}, i > j, \nexists k \in \mathcal{N} : j < k < i$. For example, "Which David Shotton's papers were featured in the dataset in 2014 but not in 2013?".

Cross-delta structured query (CD) retrieves the change-sets of a SPARQL query's results between more than one consecutive couple of versions. Formally: $CD(Q, V_i, V_j, V_m) = SD(Q, V_i, V_j) \bowtie SD(Q, V_j, V_m)$. For example, "When were articles by David Shotton added to or removed from the collection?".

Extensions of SPARQL exist to support queries on time-aware RDF datasets, that either require using non-standard languages to map data — such as τ-SPARQL [48], T-SPARQL [49], and AnQL [13] — or only works on a purpose-built database, i.e. $SPARQL^{\mathcal{T}}$ on the RDF-TX system [50]. This article proposes a methodology to support all query types on any triplestore in standard SPARQL.

In this direction, SPARQ-LTL [51] proposes a relevant approach by extending SPARQL but describing an algorithm for rewriting queries in standard SPARQL, provided that all triples are annotated with revision numbers and the revisions are accessible as named graphs. However, to the best of our knowledge, this strategy has no implementations.

2.3. Storing dynamic linked open data

This section will review existing storage and querying methodologies, focusing on supported queries, real-time operation, and generality. We consider generic a model that complies with standard RDF and can be queried via standard SPARQL on any RDF-compatible storage system.

Various archiving policies have been elaborated to store and query the evolution of RDF datasets, namely independent copies, change-based, timestamp-based, and fragment-based policies [32].

Independent copies consist of storing each version separately. It is the most straightforward model to implement and allows performing VM, SV, and CV easily. However, this approach needs a massive amount of space for storing and time for processing. Furthermore, given the different statements' versions, further diff mechanisms are required to identify what changed. Nevertheless, to date, this is the archiving policy adopted by most systems and knowledge bases, such as DBPedia [52], Wikidata [5, 53, 54], and YAGO [6].

The first version control systems for RDF was SemVersion [55], specially tailored for ontologies. It saves each version of an ontology in a separate snapshot and differences are calculated on the fly. SemVersion supports VM, SV, DM, and SD but not via SPARQL, because SPARQL became a W3C Recommendation in 2008 and SemVersion has not been updated since 2005.

The change-based policy was introduced to solve scalability problems caused by the independent copies approach. It consists of saving only the deltas between one version and the other. For this reason, DM is costless. The drawback is that additional computational costs for delta propagation are required to support version-focused queries.

The first proposal of this approach relied on a RDBMS to store the original dataset and the deltas between two consecutive versions [27]. To improve performance, deltas are pre-processed and duplicated, or unnecessary modifications are deleted. There is no support for SPARQL and queries must be formulated in SQL.

A concrete implementation of a change-based policy is R&Wbase, a version control system inspired by Git but designed for RDF [35]. Additions and deletions are stored in separate named graphs, and all queries are supported. However, this model is not fully semantic, since it requires hash tables to map revisions with change-sets. In addition, it is not triplestore-agnostic, as it supports only Fuseki and Virtuoso.

R43ples is inspired by R&WBase and perfects it by adopting a totally semantic model [34]. It is called Revi-

sion Management Ontology and records change-sets and the related provenance metadata in separate graphs using PROV-O and some new properties (e.g., `rmo:deltaAdded` and `rmo:deltaRemoved`). R43ples acts as a proxy between the data triplestore and the provenance triplestore. However, R43ples cannot be considered a generic solution, as it extends SPARQL with some keywords to simplify the queries (e.g., `REVISION`, `TAG`, `MERGE`), and the current implementation mandates using Jena TDB as the provenance triplestore.

The timestamp-based policy annotates each triple with its transaction time, that is, the timestamp of the version in which that statement was in the dataset.

x-RDF-3X is a database for RDF designed to manage high-frequency online updates, versioning, time-traversal queries, and transactions [28]. The triples are never deleted but are annotated with two fields: the insertion and deletion timestamp, where the last one has zero value for currently living versions. Afterward, updates are saved in a separate workspace and merged into various indexes at occasional savepoints. x-RDF-3X supports VM and SV queries.

v-RDFCSA uses a similar strategy but excels in reducing space requirements, compressing both the RDF archive and the timestamps attached to the triples [29]. It has basic query capabilities (VM, SV, DM, and SD on single triple patterns), and, similarly to HDT [56], SPARQL updates are not supported.

Dydra [57] is also a TB storage system, which indexes quadruples and associates their creation and addition times to form quintuples. It supports all query types but extending SPARQL with the REVISION keyword.

The fragment-based approach avoids reconstructing versions via deltas by saving only fragments of what changed. Different granularity levels are possible, depending on the requirements (a graph, a subgraph, or an entity).

Quit Store [36] inherits from R&Wbase and R43ples the Git-based distributed version control management approach. Modified fragments are saved in named graphs, and metadata are modeled according to PROV-O. The data model complies with RDF 1.1, and all query types are supported in plain SPARQL 1.1. However, the current implementation suffers from high memory requirements, as all queries are run on an in-memory quad-store.

Finally, there are hybrid storage policies that combine the changed-based approach with the timestamp-based approach. For example, OSTRICH is a triplestore that retains the first version of a dataset and subsequent deltas, as introduced in [27]. However, it merges change-sets based on timestamps to reduce redundancies between versions, adopting a change-based and timestamp-based approach simultaneously [30]. OSTRICH has native support for all types of queries.

TailR [58] also preserves the first version and succeeding diffs but reduces the computational effort to reconstruct versions by also saving some intermediate snapshots, adopting an IC/CB approach, and thus increasing space requirements. Queries are not made via SPARQL but via the Memento protocol, i.e., HTTP content-negotiation via the `Accept-Datetime` header [59].

As Section 3 will detail, the OpenCitations Data Model [60] adopts a hybrid CB/TB approach and represents provenance and changes complying with RDF. At the same time, this paper introduces time-agnostic-library, which enables all types of queries using plain SPARQL on files or any triplestores live.

3. Methodology

As discussed in section 2, Semantic Web technologies did not initially allow recording or querying change-tracking provenance. For this reason, it is necessary to adopt an external provenance model. In the context of this work, the OpenCitations Data Model (OCDM) was employed [38], summarized in Fig. 1.

According to the OCDM, a new snapshot is defined every time an entity is created or modified, and it is stored

Methodology	Storage paradigm	Queries	Live*	Generic**
SemVersion [55]	IC	VM, SV, DM, SD	+	-[d]
x-RDF-3X [28]	TB	VM, SV	-	-[a,e]
RBDMS [27]	CB	All	-	-[c,d]
R&WBase [35]	CB	All	+	-[c,e]
R43ples [34]	CB	All	+	-[b,e]
TailR [58]	IC/CB	All	+	-[d]
v-RDFCSA [29]	TB	VM, SV, DM, SD	-	-[e]
RDF-TX [50]	TB	All	-	-[b,e]
OSTRICH [30]	IC/CB/TB	All	-	-[e]
Dydra [57]	TB	All	-	-[a,b,e]
Quit Store [36]	FB	All	+	-[b]
OCDM and time-agnostic-library [60]	CB/TB	All	+	+

Table 1: Comparative between time-agnostic-library and preexisting software to perform time-agnostic queries on RDF datasets

* By "live" we mean that updates and queries can be executed in real time, without requiring pre-processing

** By "generic" we mean that provenance and change-tracking are modeled complying with RDF and can be queried via standard SPARQL on any RDF-compatible storage system

a. It extends RDF

b. It extends SPARQL

c. It requires hash tables

d. It does not use SPARQL to perform queries

e. It requires a custom or specific database

within a (provenance) named graph. The snapshots are of type `prov:Entity` and are connected to the entity described through `prov:specializationOf`. In addition, each snapshot records the validity dates (`prov:generatedAtTime`, `prov:invalidatedAtTime`), the agents responsible for creation/modification of entities' data (`prov:wasAttributedTo`), the primary sources (`prov:hasPrimarySource`), a link to the previous snapshot in time (`prov:wasDerivedFrom`), and a human readable description (`dcterms:description`).

Furthermore, OCDM extends the Provenance Ontology by introducing a new property called `oco:hasUpdateQuery`, a mechanism to record additions and deletions from an RDF graph with a SPARQL `INSERT` and SPARQL `DELETE` query string. This system makes it easier to:

- recover the current statements of an entity, as they are those available in the present dataset;
- restore an entity to a specific snapshot s_i, by applying the reverse operations of all update queries from the most recent snapshot s_n to s_{i+1}.

SPARQL update queries representing deltas must contain only absolute URIs, literals, and blank nodes, while prefixes and variables are not permitted.

From now on, we use the exemplar dataset with provenance and change-tracking information described in section 4 to introduce all the examples discussed in the following paragraphs. As shown in the Graffoo diagram [61] in Fig. 2 and Listing 1, the entity `<id/80178>`, repre-

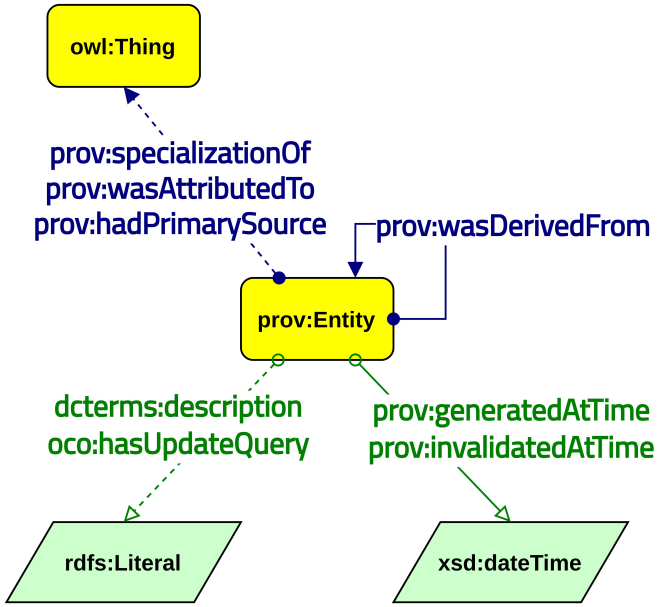

Figure 1: Provenance in the OpenCitations Data Model, represented using the graphical framework Graffoo

senting [62], is associated with the bibliographic resource `<br/86766>`, whose title is *Open access and online publishing: a new frontier in nursing?*. Moreover, `<br/86766>` cites five other resources, namely `<br/301102>`,`<br/301103>`, `<br/301104>`, `<br/301105>`, and `<br/301106>`. The identifier `<id/80178>` of `<br/86766>` was initially registered with a wrong DOI, i.e. "10.1111/j.1365-2648.2012.06023.x**.**" instead of "10.1111/j.1365-2648.2012.06023.x", where the error is in the trailing period. The agent identified by the ORCID `0000-0002-8420-0696` corrected such a mistake on October 19th, 2021, at 19:55:55. Therefore, the snapshot `<id/80178/prov/se/2>` was generated, associated with `<id/80178>`, and deriving from the previous snapshot `<id/80178/prov/se/1>`.

The taxonomy by Fernández, Polleres, and Umbrich [37] introduced in section 2.2 is used to illustrate which approaches were adopted to achieve each type of query. Therefore, a distinction is made between version materialization, delta materialization, single and cross-version structured query, single and cross-delta structured query.

3.1. Version and delta materialization

Obtaining a version materialization means retrieving the full version of a specific entity. Thus, the starting information is a resource URI and a time. Then, it is necessary to acquire the provenance information available for that entity, querying the dataset on which it is stored. In particular, the crucial data regards the existing snapshots, their generation time, and update queries expressing changes through SPARQL update query strings.

From a performance point of view, the main problem is how to get the status of a resource at a given time without reconstructing its whole history, but only the portion needed to get the result. Suppose t_n is the present state and having all the SPARQL update queries. The status of an entity at the time t_{n-k} can be obtained by adding the inverse queries in the correct order from n to $n-k+1$ and applying the queries sum to the entity's present graph.

For example, consider the graph of the entity `<id/80178>` (Fig. 2). At present, this identifier has a literal value of "10.1111/j.1365 2648.2012.06023.x". We want to determine if this value was modified recently, reconstructing the entity at time t_{n-1}. The string associated with the property `oco:hasUpdateQuery` at time t_n is shown in Fig. 2 and Listing 1.

Therefore, to reconstruct the literal value of `<id/80178>` at time t_{n-1}, it is sufficient to apply the same update query to the current graph by replacing `DELETE` with `INSERT` and `INSERT` with `DELETE`: what was deleted must be inserted, and what was inserted must be deleted to rewind the entity's time. It appears that `<id/80178>` had a different literal value at time t_{n-1}, namely "10.1111/j.1365-2648.2012.06023.x**.**". If the resource had more than two snapshots and the time of interest had been t_{n-2}, it would have been necessary to execute the same operation with the sum of the update queries associated with t_n and t_{n-1} in this order.

In addition to data, metadata related to a given change can be derived, asking for supplementary information to

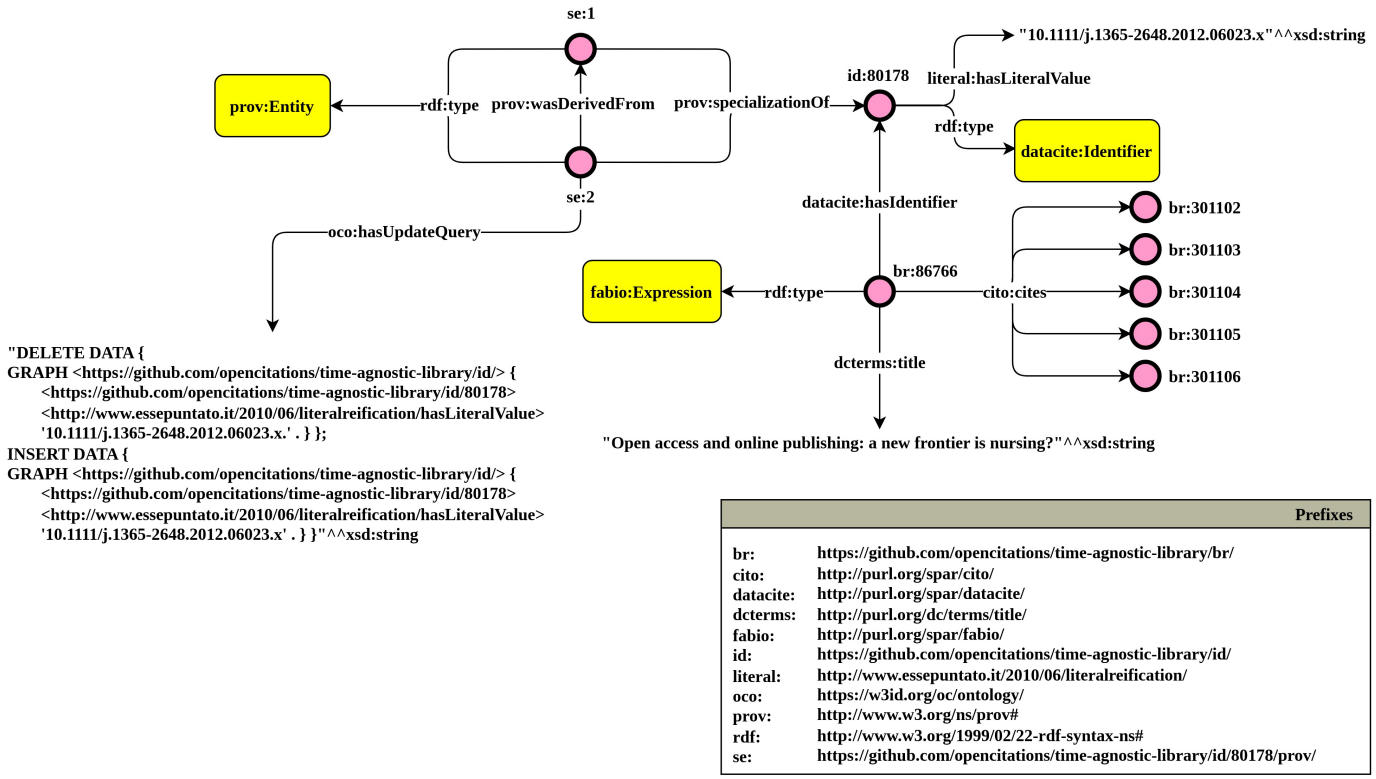

Figure 2: Usage example of the OpenCitations Data Model, shown via the graphical framework Graffoo

the provenance dataset, such as the responsible agent and the primary source. In this way, it is possible to understand who made a specific change and the information's origin. Finally, hooks to metadata related to non-reconstructed states can be returned to find out what other snapshots exist and possibly rebuild them.

The flowchart in Fig. 3 summarizes the version materialization methodology.

The process described so far is efficient in materializing a specific entity's version. However, if the goal is to obtain the history of a given resource, adopting the procedure described in Fig. 3 would mean executing, for each snapshot, all the update queries of subsequent snapshots, repeating the same update query over and over again. Since every resource graph needs to be output, it is more convenient to run the reverse update query related to each snapshot on the following snapshot graph, which was previously computed and stored.

Conversely, obtaining the materialization of a delta means returning the change between two versions. No operations are introduced in our methodology to address this operation because it is not needed since the OCDM already requires deltas to be explicitly stored as SPARQL update queries strings by adopting a change-based policy. Therefore, the diff is the starting point and is immediately available, without the need of processing provenance change tracking data to derive it. However, if more than a mere delta is required, and there is the demand to perform a single or cross-delta structured query, it is helpful to have approaches to speed up this operation, as illustrated in section 3.3.

3.2. Single and cross-version structured query

Running a structured query on versions means resolving a SPARQL query on a specific entity's snapshot, if it is a single-version query, or on multiple dataset's versions,

Algorithm 1: Version materialization

```
Function get_state_at_time(res, time, include_prov_metadata=False):
    results = execute_sparql_query(query_snapshots);
    if results is empty then
        return None, None, None;
    sort_results_by_time_descending(results);
    relevant_results = filter_results_by_time_interval(time, results);
    other_snapshots_metadata = create_empty_dictionary();
    entity_snapshots = create_empty_dictionary();
    entity_graphs = create_empty_dictionary();
    if include_prov_metadata is True then
        other_snapshots = filter_other_snapshots(results, relevant_results);
        other_snapshots_metadata = create_empty_dictionary();
        for other_snapshot in other_snapshots do
            other_snapshots_metadata[snapshot_uri] = {
            "generatedAtTime": other_snapshot[generation_time],
            "wasAttributedTo": other_snapshot[responsible_agent],
            "hadPrimarySource": other_snapshot[primary_source]
            };
    if relevant_results is empty then
        return entity_graphs, entity_snapshots, other_snapshots_metadata;
    entity_present_graph = query_dataset();
    for relevant_result in relevant_results do
        sum_update_queries = "";
        for result in results do
            if result[update_query] is not null then
                if convert_to_datetime(result[generation_time]) >
                  convert_to_datetime(relevant_result[generation_time]) then
                    sum_update_queries += result[update_query] + ";";
        execute_update_queries(entity_present_graph, sum_update_queries);
        entity_graphs[convert_to_datetime(relevant_result[1], stringify=True)] = entity_present_graph;
        entity_snapshots[relevant_result[0]] = {
        "generatedAtTime": relevant_result[generation_time],
        "wasAttributedTo": relevant_result[responsible_agent],
        "hadPrimarySource": relevant_result[primary_source]
        };
    return entity_graphs, entity_snapshots, other_snapshots_metadata;
```

```
@base <https://github.com/opencitations/time-agnostic-library/>.
@prefix cito: <http://purl.org/spar/cito/>.
@prefix datacite: <http://purl.org/spar/datacite/>.
@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix literal: <http://www.essepuntato.it/2010/06/literalreification/>.
@prefix oco: <https://w3id.org/oc/ontology/>.
@prefix prov: <http://www.w3.org/ns/prov#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.

<br/86766> a <http://purl.org/spar/fabio/Expression>;
    dcterms:title "Open access and online publishing: a new frontier in nursing?"^^xsd:string;
    cito:cites <br/301102>, <br/301103>, <br/301104>, <br/301105>, <br/301106>;
    datacite:hasIdentifier <id/80178>.

<id/80178> a datacite:Identifier;
    datacite:usesIdentifierScheme datacite:doi;
    literal:hasLiteralValue "10.1111/j.1365-2648.2012.06023.x"^^xsd:string.

<id/80178/prov/se/2> a prov:Entity;
    oco:hasUpdateQuery "
        DELETE DATA {
        GRAPH <https://github.com/opencitations/time-agnostic-library/id/> {
            <https://github.com/opencitations/time-agnostic-library/id/80178>
            <http://www.essepuntato.it/2010/06/literalreification/hasLiteralValue>
            '10.1111/j.1365-2648.2012.06023.x.' . } };
        INSERT DATA {
        GRAPH <https://github.com/opencitations/time-agnostic-library/id/> {
             <https://github.com/opencitations/time-agnostic-library/id/80178>
             <http://www.essepuntato.it/2010/06/literalreification/hasLiteralValue>
             '10.1111/j.1365-2648.2012.06023.x' . } }"^^xsd:string.
    dcterms:description "The entity 'https://github.com/opencitations/time-agnostic-library/id/80178' has been
    ↪ modified."^^xsd:string;
    prov:generatedAtTime "2021-10-19T19:55:55"^^xsd:dateTime;
    prov:specializationOf <id/80178>;
    prov:wasAttributedTo <https://orcid.org/0000-0002-8420-0696>;
    prov:wasDerivedFrom <id/80178/prov/se/1>;

<id/80178/prov/se/1> a prov:Entity;
    dcterms:description "The entity 'https://github.com/opencitations/time-agnostic-library/id/80178' has been
    ↪ created."^^xsd:string;
    prov:generatedAtTime "2021-10-10T23:44:45"^^xsd:dateTime;
    prov:hadPrimarySource <https://api.crossref.org/works/10.1007/s11192-019-03265-y>;
    prov:invalidatedAtTime "2021-10-19T19:55:55"^^xsd:dateTime;
    prov:specializationOf <id/80178>;
    prov:wasAttributedTo <https://orcid.org/0000-0002-8420-0696>.
```

Listing 1: Usage example of the OpenCitations Data Model, translated in RDF Turtle syntax

in case of a cross-version query. In both cases, a strategy must be devised to achieve the result efficiently. According to the OCDM, only deltas are stored; therefore, the dataset's past conditions must be reconstructed to query those states. However, restoring as many versions as snapshots would generate massive amounts of data, consuming time and storage. The proposed solution is to reconstruct only the past resources significant for the user's query.

Hence, given a query, the goal is to explicit all the variables, materialize every version of each entity found, and align the respective graphs temporally to execute the original query on each. To this end, the first step is to process the SPARQL query string and extract the triple patterns. Each identified triple may be joined or isolated.

Definition 4 (Joined and isolated triple pattern). A triple pattern is joined if a path exists between its subject variable and a subject IRI in the query.

Assume there are pairwise disjoint infinite sets I, V, and L (IRIs, Variables, and Literals, respectively), and assume $Path(n, n_1)$ returns $True$ if there is a route through a graph between the graph nodes n and n_1. $Joined(s, p, o) \Leftrightarrow$

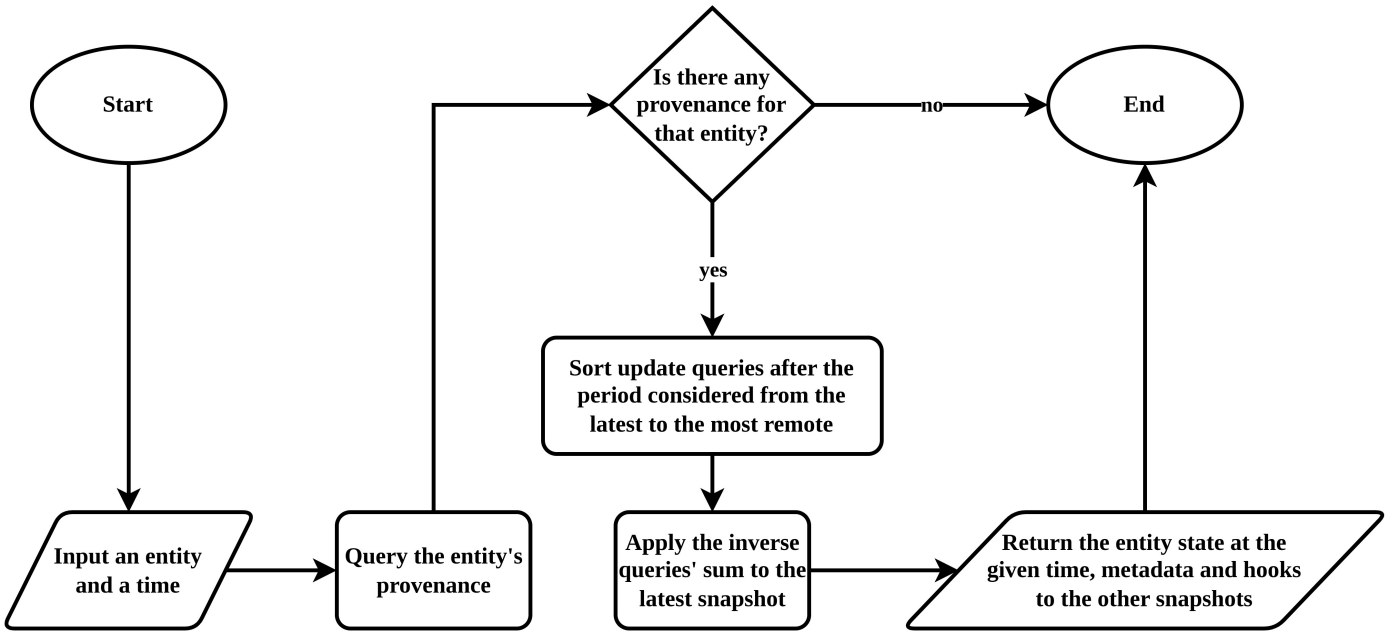

Figure 3: Flowchart illustrating the methodology to materialize an entity version at a given period

$\exists(s_1, p_1, o_1) : Path(s, s_1), s_1 \in I$, with $(s, p, o) \in (I \cup V) \times (I \cup V) \times (I \cup L \cup V)$.

Viceversa, a triple pattern is said to be isolated if such path not exists.

If a triple pattern in the input query is joined, it is possible to solve its variables using a previously reconstructed entity graph.

Consider the example in Listing 2. Once all versions of `<br/86766>` have been materialized, every possible value of the variable `?br` is known. At that point, all the possible values that `?id` had can be derived from all the URIs of `?br`. Also, the variable `?value` can be resolved similarly. It is worth noting that a variable can have different values not only in different versions but also in the same version. For instance, the bibliographical resource `<br/86766>` cites more than just another bibliographical resource (as shown in Fig. 2). Hence, `?br` takes multiple values in all of its snapshots, determining the same for `?id` and `?value`.

On the other hand, a triple pattern is isolated if it is wholly disconnected from the other patterns in the query, and its subject is a variable. The query is more generic if there are isolated triples; therefore, identifying the relevant entities is more demanding. However, if at least one URI is specified in the query, it is still possible to narrow the field so that only the strictly necessary entities are restored and not the whole dataset. Since deltas are saved as SPARQL strings, a textual search on all available deltas can be executed to find those containing the known URIs. The difference between a delta triple including all the isolated triple URIs and the isolated triple itself is equal to the relevant entities to rebuild. Listing 3 shows a time-traversal query to find all identifiers whose literal value has ever contained a trailing dot. Inside, there is the isolated triple pattern `?id literal:hasLiteralValue ?literal`, where only the predicate is known, and the subject is not explicable by other triples within the query.

Identifying all the possible values of `?id` and `?literal` at any time means discovering which nodes have ever been connected by the predicate `literal:hasLiteralValue`. This information is enclosed in the values of `oco:hasUpdateQuery` within the provenance entities' snap-

```
PREFIX literal: <http://www.essepuntato.it/2010/06/literalreification/>
PREFIX cito: <http://purl.org/spar/cito/>
PREFIX datacite: <http://purl.org/spar/datacite/>
SELECT DISTINCT ?br ?id ?value
WHERE {
    <https://github.com/opencitations/time-agnostic-library/br/86766> cito:cites ?br.
    ?br datacite:hasIdentifier ?id.
    ?id literal:hasLiteralValue ?value.
}
```

Listing 2: Example of a SPARQL query containing only joined triple patterns

```
PREFIX literal: <http://www.essepuntato.it/2010/06/literalreification/>
SELECT ?literal
WHERE {
    ?id literal:hasLiteralValue ?literal.
    FILTER REGEX(?literal, "\.$")
}
```

Listing 3: Example of a SPARQL query containing an isolated triple pattern

shots. First, the update queries including the predicate `literal:hasLiteralValue` must be isolated. Then, they have to be parsed in order to process the triples inside. All subjects and objects linked by `literal:hasLiteralValue` are reconstructed to answer the user's time-agnostic query.

It is worth mentioning that a user query can contain both joined and isolated triples. In this case, the disconnected triples are processed by carrying out textual searches on the diffs. In contrast, the connected ones are solved by recursively explicating the variables inside them, as we saw.

After detecting the relevant resources concerning the user's query, the next step depends on whether it is a single-version or a cross-version query. In the first case, for better efficiency, it is not necessary to reconstruct the whole history of every entity, but only the portion included in the input time. On the contrary, for cross-version queries, all the relevant versions of each resource must be restored. In both cases, the method adopted is the version materialization described in section 3.1.

However, the initial search cannot be answered even after all the relevant data records are obtained. Restored snapshots must be aligned to get a complete picture of events. In particular, since the property `oco:hasUpdateQuery` only records changes, if an entity was modified at time t_n, but not at t_{n+1}, that entity will appear in the t_n-related delta but not in the t_{n+1} one. The t_{n+1} graph would not include that resource, although it should be present. As a solution, entities present at time t_n but absent in the following snapshot must be copied to the t_{n+1}-related graph because they were not modified. Finally, entities' graphs are merged based on snapshots so that contemporary information is part of the same graph.

After the pre-processing described so far, performing the time-traversal query becomes a trivial task. It is sufficient to execute it on all reconstructed graphs, each associated with a snapshot relevant to that query and containing the strictly necessary information to satisfy the user's re-

quest.

The flowchart in Fig. 4 summarizes the single-version and cross-version query methodology.

3.3. Single and cross-delta structured query

Performing a structured query on deltas means focusing on change instead of the overall status of a resource. On the one hand, if the interest is limited to a specific change instance, it is called a single-delta structured query. On the other hand, if the structured query is run on the whole dataset's changes history, it is named a cross delta structured query.

Theoretically, employing the OCDM, it is possible to conduct searches on deltas without needing a dedicated library. For example, the query in Listing 4 can be used to find those identifiers whose strings have been modified. However, a similar SPARQL string requires the user to have a deep knowledge of the data model. Therefore, it is valuable to introduce a method to simplify and generalize the operation, hiding the complexity of the underlying provenance pattern.

From Listing 4, it is possible to derive two requirements: the user shall identify the entities he is interested in through a SPARQL query and specify the properties to study the change. In addition, to allow both single-delta and cross-delta structured queries, it is necessary to provide the possibility of entering a time.

Consequently, the first step is to discover the entities that respond to the user's query. One might think that it is enough to search them on the data collection and store the resources obtained. However, only the URIs currently contained in the dataset would be acquired, excluding the deleted ones. A strategy similar to that described for time-traversal queries must be implemented to satisfy the user's research across time. The query has to be pre-processed, extracting the triple patterns and recursively explicating the variables for the non-isolated ones. To this end, the past graphs of the (gradually) identified resources must be reconstructed, and the procedure is identical to the version query's one shown in Fig. 4. Likewise, if the user has input a time, only versions within that period are materialized; otherwise, all states are rebuilt. However, the difference is in the purpose because there is no need to return previous versions in this context. Rebuilding past graphs is a shortcut to explicate the query variables and identify those relevant resources in the past but not in the present dataset state. Thereby, as far as isolated triples patterns are concerned, the procedure is more streamlined. Once their URIs have been found within the update queries and the relevant entities have been stored, there is no reason to get their past conditions since they are isolated.

After all relevant entities are found, suppose a set of properties is input. In that case, the previously collected resources must be filtered, only keeping those that changed the values in the properties' set. This information can be obtained from the provenance data. On the contrary, if no predicate was indicated, it is necessary to restrict the field to those entities that have received any modification. Finally, the relevant modified entities are returned concerning the specified query, properties, and time, when they changed and how.

The flowchart in Fig. 5 summarizes the single-delta and cross-delta structured query methodology.

3.4. Implementation

We concretely implemented this methodology in a Python package, time-agnostic-library, distributed as open-source software on GitHub and Zenodo under the ISC license, archived on Software Heritage for long-term preservation, and downloadable with *pip* [63]. It makes three main classes available to the user: `AgnosticEntity`, `VersionQuery`, and `DeltaQuery`, for materializations, version queries, and delta queries. All three operations can be performed over the entire history available or by specifying a time interval via a tuple in the form `(START, END)`. In this way, each of the six retrieval functionalities considered in the taxonomy

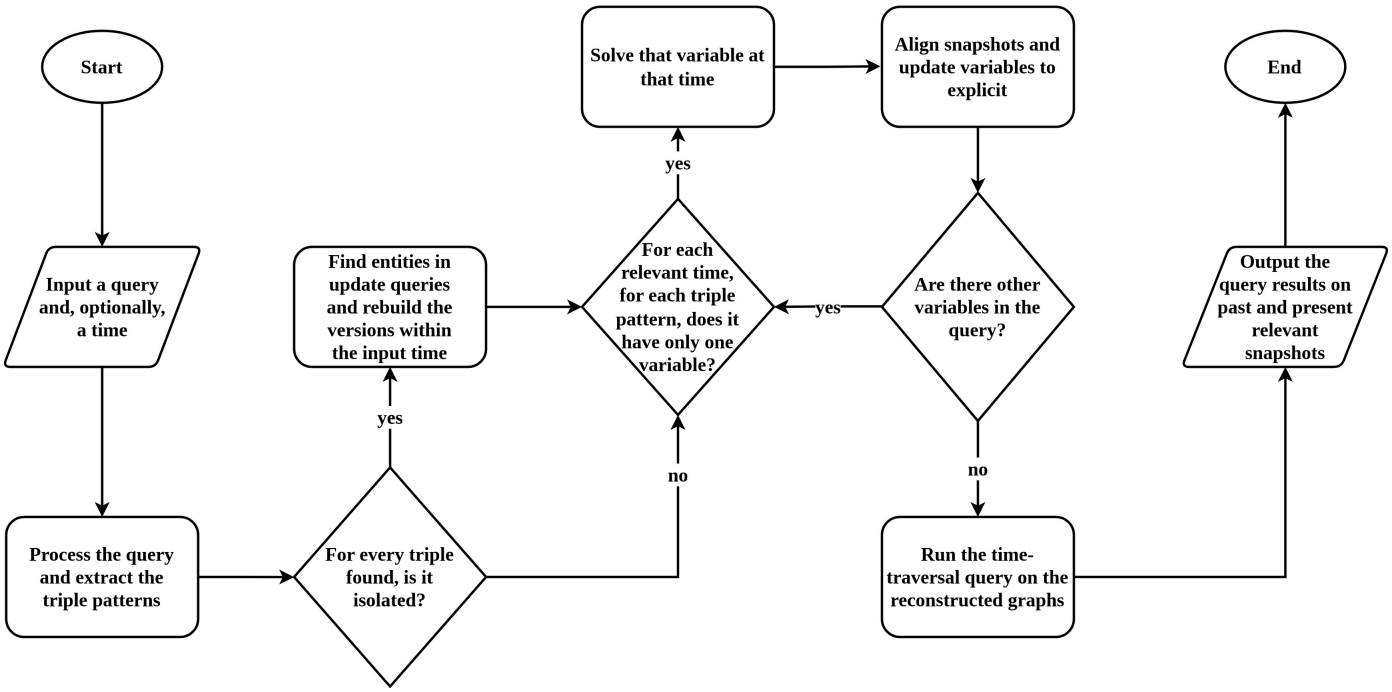

Figure 4: Flowchart illustrating the methodology to perform single-time and cross-time structured queries on versions

by Fernández et al. can be accomplished [64].

The package was tested on Blazegraph, GraphDB, Apache Jena Fuseki, and OpenLink Virtuoso, and it is fully compatible with these triplestores. In addition, time-agnostic-library provides configuration parameters to take advantage of the full-text indexing systems integrated into each of the mentioned databases to achieve instant text searches.

Moreover, the library enables one to specify the URL of a triplestore to use as a cache. The benefits of using this cache mechanism are illustrated as follows:

1. All past reconstructed graphs are saved on triplestore and never on RAM. Then, the impact of the process on the RAM is highly reduced.
2. Time-traversal queries are executed on the cache triplestore and not on graphs saved in RAM. Therefore, they are faster, as most triplestores implement optimization strategies to run queries efficiently. For example, Blazegraph uses B+Tree as a data structure, which provides search operations in logarithmic amortized time [65].
3. If a query is launched a second time, the already recovered entities' history is not reconstructed but derived from the cache.

However, the cache has three disadvantages: first, it takes up space; secondly, the current implementation does not quicken the relevant entities' discovery, and the variables must be solved each time. Third, the first execution slows down the query, as the information must be loaded into the cache triplestore.

Test-Driven Development (TDD) [66] was adopted as a software development process, and a total of 155 tests were developed, with 100% coverage. Finally, the entire test procedure was automated to ensure reproducibility, including downloading and extracting the necessary databases and starting them. For detailed documentation on the operation and use of time-agnostic-library and the cache system, see [67].

```
PREFIX datacite: <http://purl.org/spar/datacite/>
PREFIX oco: <https://w3id.org/oc/ontology/>
PREFIX prov: <http://www.w3.org/ns/prov#>
SELECT DISTINCT ?id
WHERE {
    ?se prov:specializationOf ?id; oco:hasUpdateQuery ?updateQuery.
    ?id a datacite:Identifier.
    FILTER CONTAINS (
        ?updateQuery, "http://www.essepuntato.it/2010/06/literalreification/hasLiteralValue"
    )
}
```

Listing 4: Example of a direct query on deltas

4. Evaluation and discussion

This section illustrates the quantitative evaluation we performed on time-agnostic-library through benchmarks on execution times and memory used by the various functionalities.

Before benchmarking, it was necessary to generate a dataset compliant with the OpenCitations Data Model rich in provenance information. As for the dataset content, the metadata of all the works published by the journal Scientometrics was mapped, having derived that information entirely from Crossref via its REST API [68]. It contains 4,960,087 data triples and 19,348,027 provenance triples, which correspond to 1,134,545 entities and 2,696,689 snapshots. The code to generate and modify such collections is available on GitHub [69]. Moreover, the benchmark procedure is fully reproducible and automated by running a single bash file [70].

Benchmarks are performed on a variety of triplestores, i.e., Blazegraph, GraphDB Free Edition, Apache Jena Fuseki, and OpenLink Virtuoso, and all data is available on Zenodo [71].

All the experiments were conducted using a computer with the following hardware specifications:

- CPU: Intel Core i9 12900K
- RAM: 128 GB DDR4 3200 MHz CL14
- Storage: 1 TB SSD Nvme PCIe 4.0

The benchmarks involve ten use cases: the materialization of one or all versions, single-version, single-delta, cross-version, and cross-delta structured queries containing three joined triple patterns and, finally, the same types of searches with one isolated triple pattern, as can be seen in Listing 5. Regarding queries with joined triple patterns, they refer to 20 randomly selected entities among those relevant to such queries. These entities have a variable number of provenance snapshots, ranging from a minimum of 2 to a maximum of 35. They have 20 snapshots on average with a standard deviation of 8.

Each benchmark on the execution time and RAM was performed ten times for each entity and each query type (200 executions for queries on explicit entities, 10 for the others), while the average and standard deviations were stored.

All the considered triplestores cache query results making the subsequent execution nearly instant. To solve this problem at its root, we killed each triplestore and relaunched it before each query since this cache is volatile and released when the database is closed.

Our cache system and each triplestore textual index

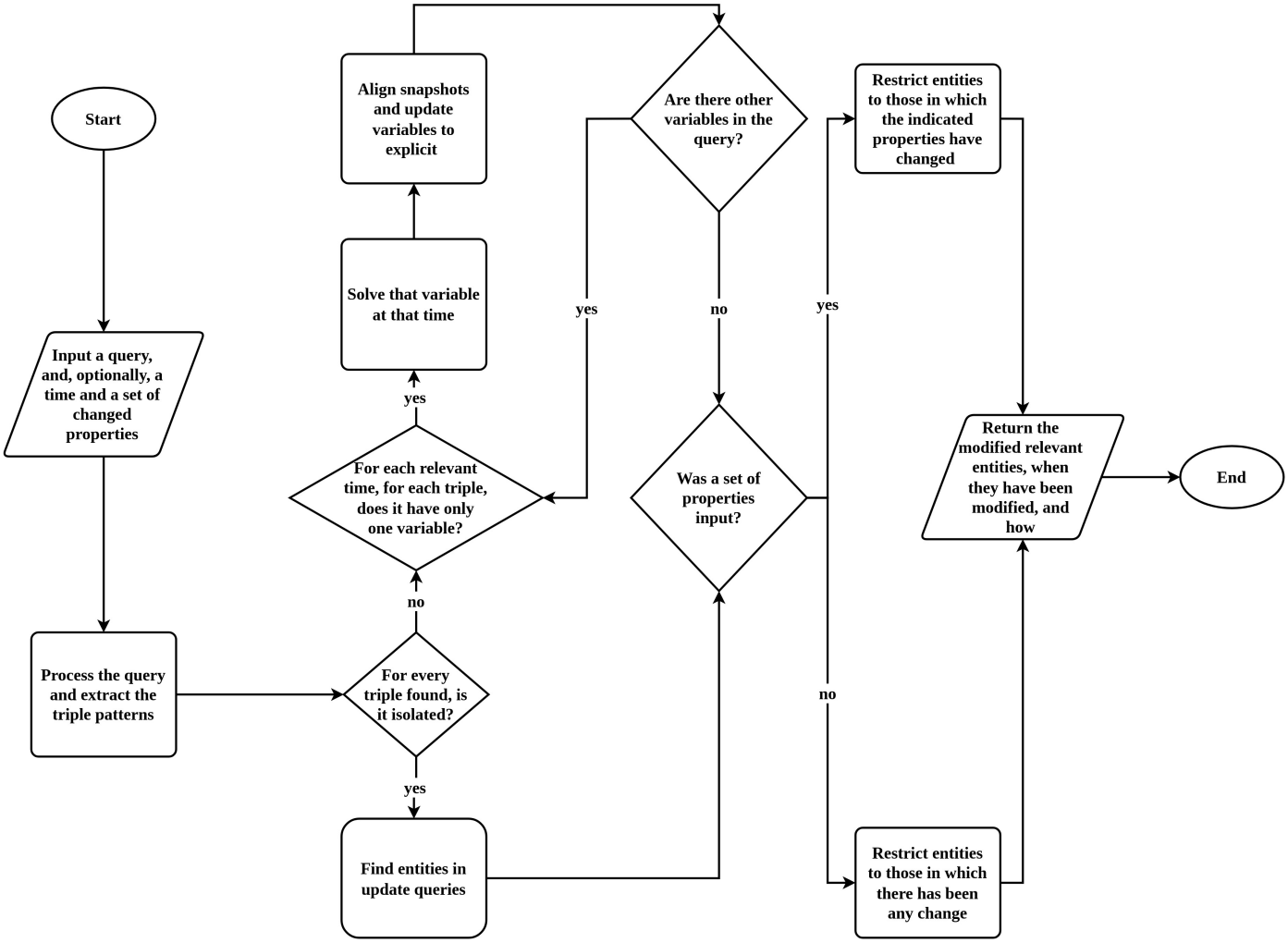

Figure 5: Flowchart illustrating the methodology to perform single-time and cross-time structured queries on deltas

were evaluated together and separately to measure their contribution to enhancing the performances. These additional features were not assessed for all the retrieval functionalities but exclusively for those that benefit from them. More precisely, the cache is employed only by those functions that involve reconstructing past graphs. On the other hand, only processes that require searching for strings within update queries take advantage of the textual indexes.

One might wonder why we used a custom benchmark and not BEAR - the main, if not the only, benchmark for time-aware datasets [37]. The reason is that BEAR does not allow experimenting with our methodology's three most relevant contributions: the possibility of performing time-traversal queries via SPARQL live. In fact, it does not support cross-version (or time-traversal) and cross-delta queries, does not use SPARQL but AnQL, and does not consider the cost of indexing the data before launching the query for those systems that do not operate in real-time.

As a baseline against which to compare results, we used the query runtime on the latest version regarding the queries on versions. This information, together with the number of snapshots involved in each query, reveals the per-snapshot overhead, i.e., the result of dividing the total runtime by the number of revisions involved in the query execution.

```
# Query on known subjects
PREFIX literal: <http://www.essepuntato.it/2010/06/literalreification/>
PREFIX cito: <http://purl.org/spar/cito/>
PREFIX datacite: <http://purl.org/spar/datacite/>
SELECT DISTINCT ?br ?id ?value
WHERE {
    <entity> cito:cites ?br.
    ?br datacite:hasIdentifier ?id.
    OPTIONAL {?id literal:hasLiteralValue ?value.}
}

# Query on unknown subjects
PREFIX datacite: <http://purl.org/spar/datacite/>
SELECT DISTINCT ?s
WHERE {
    ?s datacite:usesIdentifierScheme datacite:orcid.
}
```

Listing 5: Benchmarked queries with known and unknown subjects. In the first query, $< entity >$ is a placeholder for 20 different entities

The number of snapshots involved was calculated differently depending on the query type. Regarding version materialization, the number of snapshots is equal to the number of snapshots of that entity. Conversely, for materializations of particular versions, the number of snapshots equals the number of snapshots between the present and the version to be materialized. The same holds for cross-version and single-version queries: the snapshots' number is equal to the sum of all present and past entities' snapshots involved in the query or just the ones included between the present and the desired period.

The perspective is reversed for queries on deltas. For those, the baseline is provided by a query on the snapshots of the entities currently in the dataset. Therefore, the overhead is given by the total number of entities involved in the query, particularly those no longer present in the dataset that needs to be identified algorithmically.

Table 2 shows – in seconds – the mean and standard deviation of the time spent to complete the various operations on Blazegraph, GraphDB Free Edition, Fuseki, and Virtuoso, with and without the cache and the textual index. The values are reported with three significant figures.

By looking at the results, it can be observed that time-agnostic-library expresses the best performances for queries on known subjects. Materializing all versions took an average of 0.213s seconds on Fuseki, while a specific snapshot 0.165s seconds, with an average per-snapshot overhead of 0,0107s and 0,0211s. Conversely, on average on Fuseki, the SPARQL query on all versions took 13.5s (with 792 snapshots involved on average), on versions within a given period 1.62s seconds (15.9 updates on average), on all deltas 14.6s seconds (with 162 entities involved), and on deltas within a limited interval 2.82s seconds (for 49.2 entities).

However, such speeds are only possible if the subject is known. If it is unknown, all present and past entities relevant to explicated predicates and objects must be considered, requiring much more time. For cross-version and single-version queries, it was necessary to identify and pro-

cess 11,928 snapshots, taking 381s and 285s on average on Fuseki, while for cross-delta and single-delta queries 7642 entities solved the input query, requiring 235s and 188s on average to answer the question.

Indeed, the cache system and the textual index were implemented to reduce these timeframes as much as possible. The index alone had a marginal impact by reducing the execution of all queries up to 10 seconds, while the cache had a more significant effect, cutting alone more than half of all runtimes. Finally, combining the textual index and the cache made the results predictably the fastest in the series.

However, it is essential to highlight a drawback resulting from the cache's adoption: it improves the times only from the second execution of a given query onwards. The first time, it worsens them, involving additional write operations on the cache triplestore, as can be observed by the high standard deviation. Nevertheless, the cache always has advantages in terms of RAM, as explained below.

Table 3 shows the average RAM used by the various functionalities measured in Megabyte with three significant figures, first without and then with the cache. All operations required less than a gigabyte. The minimum was about 33 MB for materializing single versions. Conversely, the peak was about 700 MB regarding the time-traversal query with an unknown subject. Instead, the same function performed over a limited interval required up to a fourth of the memory on Blazegraph. It can be inferred that if the available RAM is insufficient, defining a period of interest helps to reduce dramatically the resources needed to answer the research.

A valid alternative to decrease RAM consumption is to use the cache system, which improves all benchmarks and cuts in a half the memory needed for time-traversal queries with isolated triple patterns. Furthermore, if the restored graphs are millions, depending on the available RAM, caching them becomes the only viable option to complete the query and avoid a crash. Additionally, even if the PC resources were sufficient, the time necessary to answer the user's query on all the past states of the dataset stored in RAM would increase exponentially with the entities involved. At the same time, a triplestore implements optimizations that allow completing this final step in a scalable way. Though, it should be noted that the cache occupies disk space. At the end of all benchmarks, the cache contains 45 dataset versions in which 10,291 entities were reconstructed, totalling 691,799 quadruples. The weight varies depending on the triplestore and is 209.7 MB for Blazegraph, 128.3 MB for GraphDB, 405.8 MB for Fuseki, and 82,1 MB for Virtuoso.

5. Conclusion

This article introduced a methodology to conduct live time-traversal queries on RDF datasets and software developed in Python implementing it. We adopted the OpenCitations Data Model to handle provenance and change tracking, which introduces a document-inspired system that stores the delta between two versions of an entity, saving the diff in a separate named graph as a SPARQL update string associated with the property `oco:hasUpdateQuery`.

Then, by analyzing existing solutions to run time-traversal queries on RDF datasets with the taxonomy by Fernández et al. [37], three requirements were established: first, all retrieval functionalities needed to be enabled; second, they had to be completed live; third, queries had to be expressed in standard SPARQL.

As far as we know, time-agnostic-library is, to date, the only one that allows performing all the time-related retrieval functionalities live via SPARQL. In addition, this software can be used for any dataset that tracks changes and provenance as described in the OCDM.

Future works include supporting all property paths in time-agnostic-library. Currently, the software only allows the inverse property path to be used. In addition, we plan to improve and extend the library's code to increase the performance of all the operations it enables, particularly

Retrieval functionality	Triplestore	w/out cache w/out index			w/out cache w/ index			w/ cache w/out index			w/ cache w/ index			Snapshots involved in the query		Query runtime on the latest snapshot	
		mean	stdev	mean per snaphot overhead	mean	stdev	mean per snaphot overhead	mean	stdev	mean per snaphot overhead	mean	stdev	mean per snaphot overhead	mean	stdev	mean	stdev
1. Materialization of all versions	**Blazegraph**	0.314s	0.0622s	0.0158s										19.9	8.36	0.158s	0.00784s
	GraphDB	1.74s	0.0715s	0.0879s										19.9	8.36	1.61s	0.0325s
	Fuseki	0.213s	0.0615s	0.0107s										19.9	8.36	0.0840s	0.00707s
	Virtuoso	0.549s	0.0637s	0.0277s										19.9	8.36	0.0310s	0.00544s
2. Materialization of a single-version	**Blazegraph**	0.270s	0.0344s	0.0346s										7.80	3.96	0.157s	0.00760s
	GraphDB	1.77s	0.0483s	0.227s										7.80	3.96	1.68s	0.0336s
	Fuseki	0.165s	0.0313s	0.0211s										7.80	3.96	0.0838s	0.00716s
	Virtuoso	0.661s	0.0344s	0.0847s										7.80	3.96	0.0311s	0.00521s
3. Cross-version structured query SP?	**Blazegraph**	16.5s	8.36s	0.0214s				13.6s	11.6s	0.0177s				792	453	0.168s	0.0116s
	GraphDB	14.5s	6.75s	0.0184s				13.2s	15.1s	0.0167s				792	453	1.71s	0.0237s
	Fuseki	13.5s	7.17s	0.0171s				19.7s	39.1s	0.0248s				792	453	0.0861s	0.00814s
	Virtuoso	15.9s	8.10s	0.0201s				9.62s	7.83s	0.0121s				792	453	0.109s	0.00463s
4. Single-version structured query SP?	**Blazegraph**	2.06s	1.14s	0.129s				1.20s	0.950s	0.0756s				15.9	10.8	0.168s	0.0134s
	GraphDB	3.22s	0.900s	0.203s				1.91s	0.959s	0.120s				15.9	10.8	1.74s	0.0228s
	Fuseki	1.62s	0.925s	0.102s				0.751s	1.13s	0.0472s				15.9	10.8	0.0851s	0.00758s
	Virtuoso	2.12s	0.953s	0.133s				0.660s	0.751s	0.0415s				15.9	10.8	0.109s	0.00441s
5. Cross-version structured query ?PO	**Blazegraph**	383s	3.95s	0.0321s	376s	1.64s	0.0315s	90.6s	65.0s	0.00759s	84.2s	67.1s	0.00706s	11928		0.246s	0.0259s
	GraphDB	375s	2.30s	0.0314s	368s	12.1s	0.0308s	111s	164s	0.00932s	107s	157s	0.00896s	11928		1.79s	0.0332s
	Fuseki	381s	2.22s	0.0320s	373s	2.64s	0.0313s	154s	297s	0.0129s	140s	285s	0.0118s	11928		0.193s	0.0116s
	Virtuoso	385s	2.26s	0.0323s	378s	3.71s	0.0317s	95.4s	122s	0.00800s	80.4s	102s	0.00674s	11928		0.0971s	0.00334s
6. Single-version structured query ?PO	**Blazegraph**	252s	2.92s	0.0447s	242s	1.24s	0.0428s	106s	87.7s	0.0187s	97.7s	85.6s	0.0173s	5651		0.228s	0.0130s
	GraphDB	231s	1.47s	0.0409s	230s	3.01s	0.0407s	99.3s	92.9s	0.0176s	95.8s	88.3s	0.0170s	5651		1.78s	0.0168s
	Fuseki	235s	1.81s	0.0416s	229s	1.64s	0.0405s	117s	147s	0.0208s	109s	147s	0.0193s	5651		0.204s	0.0135s
	Virtuoso	251s	1.45s	0.0444s	246s	2.34s	0.0436s	83.5s	81.9s	0.0148s	96.8s	74.3s	0.0171s	5651		0.102s	0.00362s
7. Cross-delta structured query SP?	**Blazegraph**	17.9s	9.19s	0.110s										162	86.5	0.189s	0.0177s
	GraphDB	15.4s	7.11s	0.0951s										162	86.5	1.79s	0.0296s
	Fuseki	14.6s	7.80s	0.0903s										162	86.5	0.108s	0.0169s
	Virtuoso	17.8s	9.07s	0.110s										162	86.5	0.474s	0.00923s
8. Single-delta structured query SP?	**Blazegraph**	3.31s	1.93s	0.0673s										49.1	30.2	0.188s	0.0189s
	GraphDB	4.44s	1.64s	0.0904s										49.1	30.2	1.80s	0.0257s
	Fuseki	2.82s	1.71s	0.0574s										49.1	30.2	0.108s	0.0181s
	Virtuoso	4.04s	1.88s	0.0823s										49.1	30.2	0.474s	0.00869s
9. Cross-delta structured query ?PO	**Blazegraph**	225s	5.91s	0.0295s	218s	5.15s	0.0285s							7642		0.332s	0.0257s
	GraphDB	206s	1.22s	0.0270s	204s	1.51s	0.0266s							7642		1.84s	0.0301s
	Fuseki	208s	3.46s	0.0272s	205s	3.62s	0.0268s							7642		0.189s	0.00793s
	Virtuoso	235s	1.89s	0.0308s	214s	16.9s	0.0279s							7642		0.445s	0.0113s
10. Single-delta structured query ?PO	**Blazegraph**	203s	1.17s	0.0266s	195s	1.43s	0.0255s							7642		0.340s	0.0306s
	GraphDB	189s	2.64s	0.0248s	180s	1.38s	0.0236s							7642		1.85s	0.0280s
	Fuseki	188s	1.97s	0.0246s	179s	2.55s	0.0234s							7642		0.200s	0.0214s
	Virtuoso	207s	10.8s	0.0270s	198s	3.90s	0.0259s							7642		0.453s	0.00665s

Table 2: Mean time in seconds spent to complete the various operations on Blazegraph, GraphDB Free Edition, Fuseki, and Virtuoso, with and without the cache and the textual index. The values are reported with three significant figures

Retrieval functionality	Triplestore	w/out cache w/out index		w/ cache w/out index	
		mean	stdev	mean	stdev
1. Materialization of all versions	**Blazegraph**	43.0MB	7.29MB		
	GraphDB	42.9MB	7.23MB		
	Fuseki	43.1MB	7.21MB		
	Virtuoso	43.1MB	7.23MB		
2. Materialization of a single-version	**Blazegraph**	33.2MB	0.546MB		
	GraphDB	33.2MB	0.543MB		
	Fuseki	33.4MB	0.536MB		
	Virtuoso	33.4MB	0.570MB		
3. Cross-version structured query SP?	**Blazegraph**	244MB	129MB	153MB	79.1MB
	GraphDB	245MB	126MB	154MB	79.1MB
	Fuseki	244MB	127MB	155MB	78.1MB
	Virtuoso	248MB	127MB	101MB	71.8MB
4. Single-version structured query SP?	**Blazegraph**	45.9MB	6.54MB	35.4MB	1.78MB
	GraphDB	46.4MB	6.70MB	35.4MB	1.78MB
	Fuseki	46.3MB	6.64MB	35.4MB	1.76MB
	Virtuoso	46.5MB	6.76MB	36.7MB	4.46MB
5. Cross-version structured query ?PO	**Blazegraph**	707MB	100MB	300MB	129MB
	GraphDB	677MB	44.8MB	304MB	127MB
	Fuseki	665MB	7.99MB	303MB	127MB
	Virtuoso	662MB	6.08MB	299MB	117MB
6. Single-version structured query ?PO	**Blazegraph**	181MB	1.35MB	147MB	14.9MB
	GraphDB	186MB	1.83MB	148MB	15.3MB
	Fuseki	185MB	3.31MB	149MB	15.8MB
	Virtuoso	185MB	0.440MB	161MB	22.3MB
7. Cross-delta structured query SP?	**Blazegraph**	247MB	129MB		
	GraphDB	239MB	128MB		
	Fuseki	244MB	128MB		
	Virtuoso	247MB	128MB		
8. Single-delta structured query SP?	**Blazegraph**	49.1MB	7.23MB		
	GraphDB	49.5MB	7.33MB		
	Fuseki	49.4MB	7.31MB		
	Virtuoso	49.8MB	7.40MB		
9. Cross-delta structured query ?PO	**Blazegraph**	73.3MB	0.261MB		
	GraphDB	74.3MB	0.206MB		
	Fuseki	74.3MB	0.144MB		
	Virtuoso	74.2MB	0.224MB		
10. Single-delta structured query ?PO	**Blazegraph**	71.7MB	0.301MB		
	GraphDB	72.9MB	0.250MB		
	Fuseki	72.9MB	0.322MB		
	Virtuoso	72.8MB	0.232MB		

Table 3: Mean RAM used by the various functionalities measured in Megabyte, first without and then with the cache. The data are reported with three significant figures

in running structured queries involving isolated triple patterns.

Finally, we aim to use time-agnostic-library to address specific needs derived from OpenCitations' use cases. These include a system to enable users to understand how and why an entity was modified in time and involve domain

experts in the curatorship of data while keeping track of the changes and their responsible agents.

Acknowledgements

This work has been partially funded by the European Union's Horizon 2020 research and innovation program under grant agreement No 101017452 (OpenAIRE-Nexus Project). We want to thank the responsible editor of the previous version of this article (available at `https://doi.org/10.48550/arXiv.2210.02534` and submitted initially to the Semantic Web Journal), Katja Hose, Ruben Taelman, and all the other anonymous reviewers of that version for their tremendous contribution to the improvement of this article. Their open reviews are available at `https://www.semantic-web-journal.net/content/performing-live-time-traversal-queries-rdf-datasets`. Our rebuttal letter to their comments, which we have addressed in the present version, is available at `https://doi.org/10.5281/zenodo.7162279`. We want to thank Fabio Vitali for the constructive feedback, and Simone Persiani, for the valuable guidance throughout the use of the Python library *oc_ocdm*. We also thank Silvia Di Pietro for the language editing and proofreading

References

[1] S. Garfinkel, Wikipedia and the Meaning of Truth, MIT Technology Review (2008).
URL `https://stephencodrington.com/Blogs/Hong_Kong_Blog/Entries/2009/4/11_What_is_Truth_files/Wikipedia%20and%20the%20Meaning%20of%20Truth.pdf`

[2] M.-R. Koivunen, E. Miller, Semantic Web Activity, edition: W3C Volume: 11 02 (Nov. 2001).
URL `https://www.w3.org/2001/12/semweb-fin/w3csw`

[3] T. Käfer, A. Abdelrahman, J. Umbrich, P. O'Byrne, A. Hogan, Observing Linked Data Dynamics, in: D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, P. Cimiano, O. Corcho, V. Presutti, L. Hollink, S. Rudolph (Eds.), The Semantic Web: Semantics and Big Data, Vol. 7882, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 213–227, series Title: Lecture Notes in Computer Science. `doi:10.1007/978-3-642-38288-8\_15`.

[4] F. Orlandi, A. Passant, Modelling provenance of DBpedia resources using Wikipedia contributions, Journal of Web Semantics 9 (2) (2011) 149–164. `doi:10.1016/j.websem.2011.03.002`.
URL `https://linkinghub.elsevier.com/retrieve/pii/S1570826811000175`

[5] P. Dooley, B. Božić, Towards Linked Data for Wikidata Revisions and Twitter Trending Hashtags, in: Proceedings of the 21st International Conference on Information Integration and Web-based Applications & Services, ACM, Munich Germany, 2019, pp. 166–175. `doi:10.1145/3366030.3366048`.
URL `https://dl.acm.org/doi/10.1145/3366030.3366048`

[6] Y. Project, Download data, code, and logo of Yago projects (2021).
URL `https://yago-knowledge.org/downloads`

[7] J. Umbrich, M. Hausenblas, A. Hogan, A. Polleres, S. Decker, Towards Dataset Dynamics: Change Frequency of Linked Open Data Sources, in: C. Bizer, T. Heath, T. Berners-Lee, M. Hausenblas (Eds.), Proceedings of the WWW2010 Workshop on Linked Data on the Web, CEUR Workshop Proceedings, Raleigh, USA, 2010, pp. 73–81.
URL `http://ceur-ws.org/Vol-628/ldow2010_paper12.pdf`

[8] J. J. Carroll, C. Bizer, P. Hayes, P. Stickler, Named graphs, provenance and trust, in: Proceedings of the 14th international conference on World Wide Web - WWW '05, ACM Press, Chiba, Japan, 2005, p. 613. `doi:10.1145/1060745.1060835`.
URL `http://portal.acm.org/citation.cfm?doid=1060745.1060835`

[9] P. Pediaditis, G. Flouris, I. Fundulaki, V. Christophides, On Explicit Provenance Management in RDF/S Graphs, in: First Workshop on the Theory and Practice of Provenance, USENIX, San Francisco, CA, USA, 2009, pp. 1–10.
URL `https://www.usenix.org/legacy/event/tapp09/tech/full_papers/pediaditis/pediaditis.pdf`

[10] G. Flouris, I. Fundulaki, P. Pediaditis, Y. Theoharis, V. Christophides, Coloring RDF Triples to Capture Provenance, in: D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, A. Bernstein, D. R. Karger, T. Heath, L. Feigenbaum, D. Maynard, E. Motta, K. Thirunarayan (Eds.), The Semantic Web - ISWC 2009, Vol. 5823, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 196–212, series Title: Lecture Notes in Computer Science. `doi:10.1007/978-3-642-04930-9\_13`.
URL `http://link.springer.com/10.1007/978-3-642-04930`

-9_13

[11] T. Berners-Lee, Notation 3 Logic (Aug. 2005).
URL https://www.w3.org/DesignIssues/N3Logic

[12] R. Dividino, S. Sizov, S. Staab, B. Schueler, Querying for provenance, trust, uncertainty and other meta knowledge in RDF, Journal of Web Semantics 7 (3) (2009) 204–219. doi:10.1016/j.websem.2009.07.004.
URL https://linkinghub.elsevier.com/retrieve/pii/S1570826809000237

[13] A. Zimmermann, N. Lopes, A. Polleres, U. Straccia, A general framework for representing, reasoning and querying with annotated Semantic Web data, Journal of Web Semantics 11 (2012) 72–95. doi:10.1016/j.websem.2011.08.006.
URL https://linkinghub.elsevier.com/retrieve/pii/S1570826811000771

[14] J. Hoffart, F. M. Suchanek, K. Berberich, G. Weikum, YAGO2: A spatially and temporally enhanced knowledge base from Wikipedia, Artificial Intelligence 194 (2013) 28–61. doi:10.1016/j.artint.2012.06.001.
URL https://linkinghub.elsevier.com/retrieve/pii/S0004370212000719

[15] O. Hartig, B. Thompson, Foundations of an Alternative Approach to Reification in RDF, arXiv:1406.3399 [cs]ArXiv: 1406.3399 (Mar. 2019).
URL http://arxiv.org/abs/1406.3399

[16] S. S. Sahoo, O. Bodenreider, P. Hitzler, A. Sheth, K. Thirunarayan, Provenance Context Entity (PaCE): Scalable Provenance Tracking for Scientific RDF Data, in: D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, M. Gertz, B. Ludäscher (Eds.), Scientific and Statistical Database Management, Vol. 6187, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 461–470, series Title: Lecture Notes in Computer Science. doi:10.1007/978-3-642-13818-8_32.

[17] V. Nguyen, O. Bodenreider, A. 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, ACM Press, Seoul, Korea, 2014, pp. 759–770. doi:10.1145/2566486.2567973.
URL http://dl.acm.org/citation.cfm?doid=2566486.2567973

[18] E. Damiani, B. Oliboni, E. Quintarelli, L. Tanca, A graph-based meta-model for heterogeneous data management, Knowledge and Information Systems 61 (1) (2019) 107–136. doi:10.1007/s10115-018-1305-8.
URL http://link.springer.com/10.1007/s10115-018-1305-8

[19] F. M. Suchanek, J. Lajus, A. Boschin, G. Weikum, Knowledge Representation and Rule Mining in Entity-Centric Knowledge Bases, in: M. Krötzsch, D. Stepanova (Eds.), Reasoning Web. Explainable Artificial Intelligence, Vol. 11810, Springer International Publishing, Cham, 2019, pp. 110–152, series Title: Lecture Notes in Computer Science. doi:10.1007/978-3-030-31423-1_4.
URL http://link.springer.com/10.1007/978-3-030-31423-1_4

[20] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers, B. Plale, Y. Simmhan, E. Stephan, J. V. den Bussche, The Open Provenance Model core specification (v1.1), Future Generation Computer Systems 27 (6) (2011) 743–756. doi:10.1016/j.future.2010.07.005.
URL https://linkinghub.elsevier.com/retrieve/pii/S0167739X10001275

[21] P. P. da Silva, D. L. McGuinness, R. Fikes, A proof markup language for Semantic Web services, Information Systems 31 (4-5) (2006) 381–395. doi:10.1016/j.is.2005.02.003.
URL https://linkinghub.elsevier.com/retrieve/pii/S0306437905000281

[22] T. Lebo, S. Sahoo, D. McGuinness, PROV-O: The PROV Ontology, place: PROV-O Volume: 04 30 (Apr. 2013).
URL http://www.w3.org/TR/2013/REC-prov-o-20130430/

[23] S. S. Sahoo, A. P. Sheth, Provenir Ontology: Towards a Framework for eScience Provenance Management (Oct. 2009).
URL https://corescholar.libraries.wright.edu/knoesis/80

[24] P. Caplan, Understanding PREMIS: an overview of the PREMIS Data Dictionary for Preservation Metadata (2017).
URL https://www.loc.gov/standards/premis/understanding-premis-rev2017.pdf

[25] P. Ciccarese, E. Wu, G. Wong, M. Ocana, J. Kinoshita, A. Ruttenberg, T. Clark, The SWAN biomedical discourse ontology, Journal of Biomedical Informatics 41 (5) (2008) 739–751. doi:10.1016/j.jbi.2008.04.010.
URL https://linkinghub.elsevier.com/retrieve/pii/S1532046408000580

[26] D. U. Board, DCMI Metadata Terms (Jan. 2020).
URL http://dublincore.org/specifications/dublin-core/dcmi-terms/2020-01-20/

[27] D.-H. Im, S.-W. Lee, H.-J. Kim, A Version Management Framework for RDF Triple Stores, International Journal of Software Engineering and Knowledge Engineering 22 (01) (2012) 85–106. doi:10.1142/S0218194012500040.
URL https://www.worldscientific.com/doi/abs/10.1142/S0

218194012500040

[28] T. Neumann, G. Weikum, x-RDF-3X: Fast Querying, High Update Rates, and Consistency for RDF Databases, Proceedings of the VLDB Endowment 3 (2010) 256–263.

[29] A. Cerdeira-Pena, A. Farina, J. D. Fernandez, M. A. Martinez-Prieto, Self-Indexing RDF Archives, in: 2016 Data Compression Conference (DCC), IEEE, Snowbird, UT, USA, 2016, pp. 526–535. doi:10.1109/DCC.2016.40.
URL http://ieeexplore.ieee.org/document/7786197/

[30] R. Taelman, M. Vander Sande, J. Van Herwegen, E. Mannens, R. Verborgh, Triple storage for random-access versioned querying of rdf archives, Journal of Web Semantics 54 (2019) 4–28. doi:https://doi.org/10.1016/j.websem.2018.08.001.

[31] T. Pellissier Tanon, F. Suchanek, Querying the Edit History of Wikidata, in: P. Hitzler, S. Kirrane, O. Hartig, V. de Boer, M.-E. Vidal, M. Maleshkova, S. Schlobach, K. Hammar, N. Lasierra, S. Stadtmüller, K. Hose, R. Verborgh (Eds.), The Semantic Web: ESWC 2019 Satellite Events, Vol. 11762, Springer International Publishing, Cham, 2019, pp. 161–166, series Title: Lecture Notes in Computer Science. doi:10.1007/978-3-030-32327-1_32.

[32] O. Pelgrin, L. Galárraga, K. Hose, Towards fully-fledged archiving for RDF datasets, Semantic Web Journal 12 (6) (2021) 903–925. doi:10.3233/SW-210434.

[33] N. Noy, M. Musen, Promptdiff: A Fixed-Point Algorithm for Comparing Ontology Versions, in: Proc. of IAAI, 2002, pp. 744–750.

[34] M. Graube, S. Hensel, L. Urbas, Open semantic revision control with r43ples: Extending sparql to access revisions of named graphs, in: Proceedings of the 12th International Conference on Semantic Systems, 2016, pp. 49–56.

[35] M. Sande, P. Colpaert, R. Verborgh, S. Coppens, E. Mannens, R. Walle, R&Wbase: Git for triples, in: Proceedings of the 6th Workshop on Linked Data on the Web. 996. CEUR Workshop Proceedings, 2013, pp. 1–5.

[36] N. Arndt, P. Naumann, N. Radtke, M. Martin, E. Marx, Decentralized collaborative knowledge management using git, Journal of Web Semantics 54 (2019) 29–47. doi:https://doi.org/10.1016/j.websem.2018.08.002.

[37] J. D. Fernández, J. Umbrich, A. Polleres, M. Knuth, Evaluating query and storage strategies for RDF archives, Semantic Web 10 (2) (2019) 247–291. doi:10.3233/SW-180309.
URL https://doi.org/10.3233/SW-180309

[38] M. Daquino, S. Peroni, D. Shotton, The OpenCitations Data Model, artwork Size: 836876 Bytes Publisher: figshare (2020). doi:10.6084/M9.FIGSHARE.3443876.V7.
URL https://figshare.com/articles/online_resource/Metadata_for_the_OpenCitations_Corpus/3443876/7

[39] L. F. Sikos, D. Philp, Provenance-Aware Knowledge Representation: A Survey of Data Models and Contextualized Knowledge Graphs, Data Science and Engineering 5 (3) (2020) 293–316. doi:10.1007/s41019-020-00118-0.
URL https://link.springer.com/10.1007/s41019-020-00118-0

[40] F. Manola, E. Miller, RDF Primer (Feb. 2004).
URL http://www.w3.org/TR/2004/REC-rdf-primer-20040210/

[41] D. Beckett, RDF Syntaxes 2.0 (Apr. 2010).
URL https://www.w3.org/2009/12/rdf-ws/papers/ws11

[42] W3C, Defining N-ary Relations on the Semantic Web (Dec. 2006).
URL http://www.w3.org/TR/2006/NOTE-swbp-n-aryRelations-20060412/

[43] P. Groth, A. Gibson, J. Velterop, The anatomy of a nanopublication, Information Services & Use 30 (1-2) (2010) 51–56. doi:10.3233/ISU-2010-0613.
URL https://www.medra.org/servlet/aliasResolver?alias=iospress&doi=10.3233/ISU-2010-0613

[44] O. Udrea, D. R. Recupero, V. S. Subrahmanian, Annotated RDF, ACM Transactions on Computational Logic 11 (2) (2010) 1–41. doi:10.1145/1656242.1656245.
URL https://dl.acm.org/doi/10.1145/1656242.1656245

[45] Y. Gil, J. Cheney, P. Groth, O. Hartig, S. Miles, L. Moreau, P. Silva, Provenance XG Final Report, type: W3C. (Dec. 2010).
URL http://www.w3.org/2005/Incubator/prov/XGR-prov-20101214/

[46] A. Gschwend, O. Lassila, PROPOSED RDF-star working group charter (2022).
URL https://w3c.github.io/rdf-star-wg-charter/

[47] PROV-DM: The PROV data model (2013).
URL http://www.w3.org/TR/2013/REC-prov-dm-20130430/

[48] J. Tappolet, A. Bernstein, Applied temporal RDF: Efficient temporal querying of RDF data with SPARQL, in: L. Aroyo, P. Traverso, F. Ciravegna, P. Cimiano, T. Heath, E. Hyvönen, R. Mizoguchi, E. Oren, M. Sabou, E. Simperl (Eds.), The Semantic Web: Research and Applications, Vol. 5554, Springer Berlin Heidelberg, 2009, pp. 308–322, series Title: Lecture Notes in Computer Science. doi:10.1007/978-3-642-02121-3_25.

[49] F. Grandi, T-sparql: A tsql2-like temporal query language for rdf., in: ADBIS (local proceedings), 2010, pp. 21–30.

[50] C. Zaniolo, S. Gao, M. Atzori, M. Chen, J. Gu, User-friendly temporal queries on historical knowledge bases, Information and Computation 259 (2018) 444–459. doi:10.1016/j.ic.2017.08.012.

[51] V. Fionda, M. W. Chekol, G. Pirrò, Gize: A time warp in the

web of data, in: SEMWEB, 2016.

[52] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, C. Bizer, DBpedia – A large-scale, multilingual knowledge base extracted from Wikipedia, Semantic Web 6 (2) (2015) 167–195. doi:10.3233/SW-140134.
URL https://www.medra.org/servlet/aliasResolver?alias=iospress&doi=10.3233/SW-140134

[53] F. Erxleben, M. Günther, M. Krötzsch, J. Mendez, D. Vrandečić, Introducing Wikidata to the Linked Data Web, in: The Semantic Web – ISWC 2014, Springer International Publishing, 2014, pp. 50–65.

[54] Wikidata:Database download (Jun. 2021).
URL https://www.wikidata.org/wiki/Wikidata:Database_download

[55] M. Völkel, W. Winkler, Y. Sure, S. Kruk, M. Synak, SemVersion: A Versioning System for RDF and Ontologies, in: Proc. of ESWC, 2005.

[56] J. D. Fernández, M. A. Martínez-Prieto, C. Gutiérrez, A. Polleres, M. Arias, Binary RDF representation for publication and exchange (HDT), Journal of Web Semantics 19 (2013) 22–41. doi:https://doi.org/10.1016/j.websem.2013.01.002.

[57] J. Anderson, RDF graph stores as convergent datatypes, in: Companion Proceedings of The 2019 World Wide Web Conference, ACM, 2019, pp. 940–942. doi:10.1145/3308560.3316517.

[58] P. Meinhardt, M. Knuth, H. Sack, Tailr: A platform for preserving history on the web of data, in: Proceedings of the 11th International Conference on Semantic Systems, SEMANTICS '15, Association for Computing Machinery, New York, NY, USA, 2015, p. 57–64. doi:10.1145/2814864.2814875.

[59] S. M. Jones, M. Klein, H. V. d. Sompel, M. L. Nelson, M. C. Weigle, Interoperability for accessing versions of web resources with the memento protocol, in: D. Gomes, E. Demidova, J. Winters, T. Risse (Eds.), The Past Web: Exploring Web Archives, Springer International Publishing, 2021, pp. 101–126. doi:10.1007/978-3-030-63291-5_9.

[60] I. Heibi, S. Peroni, D. Shotton, Software review: COCI, the OpenCitations Index of Crossref open DOI-to-DOI citations, Scientometrics 121 (2) (2019) 1213–1228. doi:10.1007/s11192-019-03217-6.
URL http://link.springer.com/10.1007/s11192-019-03217-6

[61] R. Falco, A. Gangemi, S. Peroni, D. Shotton, F. Vitali, Modelling OWL Ontologies with Graffoo, in: V. Presutti, E. Blomqvist, R. Troncy, H. Sack, I. Papadakis, A. Tordai (Eds.), The Semantic Web: ESWC 2014 Satellite Events, Vol. 8798, Springer International Publishing, Cham, 2014, pp. 320–325, series Title: Lecture Notes in Computer Science. doi:10.1007/978-3-319-11955-7_42.
URL http://link.springer.com/10.1007/978-3-319-11955-7_42

[62] R. Watson, M. Cleary, D. Jackson, G. E. Hunt, Open access and online publishing: a new frontier in nursing?: Editorial, Journal of Advanced Nursing 68 (9) (2012) 1905–1908. doi:10.1111/j.1365-2648.2012.06023.x.
URL https://onlinelibrary.wiley.com/doi/10.1111/j.1365-2648.2012.06023.x

[63] A. Massari, time-agnostic-library (Sep. 2022).
URL https://archive.softwareheritage.org/swh:1:snp:3dbff3452c2e3ed0b96c3de75a27a973c262b0ff

[64] J. Fernández, A. Polleres, J. Umbrich, Towards Efficient Archiving of Dynamic Linked, in: DIACRON@ESWC, Computer Science, Portorož, Slovenia, 2015, pp. 34–49.

[65] L. SYSTAP, The bigdata® RDF Database (May 2013).
URL https://blazegraph.com/docs/bigdata_architecture_whitepaper.pdf

[66] K. Beck, Test-driven development: by example, The Addison-Wesley signature series, Addison-Wesley, Boston, 2003.

[67] A. Massari, Documentation of time-agnostic-library software (Sep. 2022). doi:10.5281/zenodo.7046272.

[68] G. Hendricks, D. Tkaczyk, J. Lin, P. Feeney, Crossref: The sustainable source of community-owned scholarly metadata, Quantitative Science Studies 1 (1) (2020) 414–427. doi:10.1162/qss_a_00022.
URL https://direct.mit.edu/qss/article/1/1/414-427/15577

[69] A. Massari, time_agnostic (Sep. 2021).
URL https://archive.softwareheritage.org/swh:1:snp:a4870cfd8555201cc8de64193cbb283758873660

[70] A. Massari, time-agnostic-library: benchmarks on execution times and memory, version Number: 3.0.0 Type: software (2022). doi:10.5281/ZENODO.5549648.

[71] A. Massari, time-agnostic-library: test datasets (May 2022). doi:10.5281/zenodo.6399069.