Toward RDF Normalization

Regina Ticona-Herrera[1,3], Joe Tekli[2], Sébastien Laborie[1],
Irvin Dongo[1], Renato Guzman[3], and Richard Chbeir[1]

[1] University of Pau - LIUPPA, Anglet, France
[2] Lebanese American University, Byblos, Lebanon
[3] San Pablo Catholic University, Arequipa, Peru
{reginapaola.ticonaherrera, sebastien.laborie, irvin.dongo}@univ-pau.fr,
joe.tekli@lau.edu.lb, renato.guzman@ucsp.edu.pe, rchbeir@acm.org

Abstract. Billions of RDF triples are currently available on the Web through the Linked Open Data cloud (e.g., DBpedia, LinkedGeoData and New York Times). Governments, universities as well as companies (e.g., BBC, CNN) are also producing huge collections of RDF triples and exchanging them through different serialization formats (e.g., RDF/XML, Turtle, N-Triple, etc.). However, RDF descriptions (i.e., graphs and serializations) are verbose in syntax, often contain redundancies, and could be generated differently even when describing the same resources, which would have a negative impact on various RDF-based applications (e.g., RDF storage, processing time, loading time, similarity measuring, mapping, alignment, and versioning). Hence, to improve RDF processing, we propose here an approach to clean and eliminate redundancies from such RDF descriptions as a means of transforming different descriptions of the same information into one representation, which can then be tuned, depending on the target application (information retrieval, compression, etc.). Experimental tests show significant improvements, namely in reducing RDF description loading time and file size.

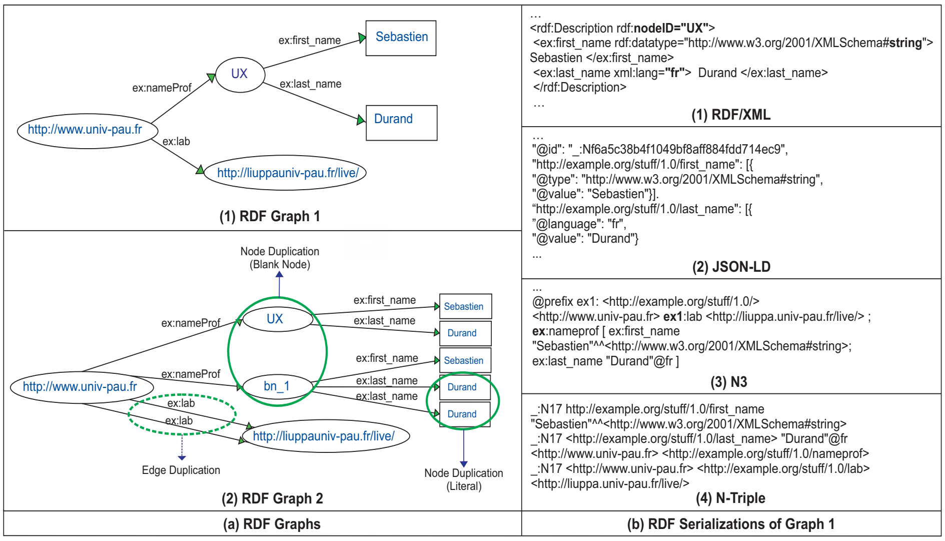
Keywords: RDF Graph, Serialization, Redundancies and Disparities

1 Introduction

Since several years, the Web has evolved from a Web of linked documents to a Web of linked data. As a result, a huge amount of RDF statements, in the form of $< Subject, Predicate, Object >$ triples, are currently available online through the Linked Open Data cloud thanks to different projects[1](e.g., *DBpedia*, *LinkedGeoData*, *New York Times*, etc.). These triples are usually stored in RDF datasets after being serialized into several machine readable formats such as RDF/XML, N-Triple, Turtle, N3 or JSON-LD. Therefore, RDF can be described in different ways as shown in Fig. 1 where an RDF description is represented by a graph (cf. Fig. 1.a) and several RDF formats (serializations) (Figure 1.b).

However, in different scenarios (e.g., collaborative RDF graph generation [6], automatic RDF serialization [12,16], etc.), RDF descriptions might be verbose

[1] `http://linkedgeodata.org`, `http://data.nytimes.com/`, `http://dbpedia.org`



```
…
<rdf:Description rdf:nodeID="UX">
 <ex:first_name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
Sebastien </ex:first_name>
 <ex:last_name xml:lang="fr">  Durand </ex:last_name>
 </rdf:Description>
 …
```

(1) RDF/XML

```
…
"@id": "_:Nf6a5c38b4f1049bf8aff884fdd714ec9",
"http://example.org/stuff/1.0/first_name": [{
"@type": "http://www.w3.org/2001/XMLSchema#string",
"@value": "Sebastien"}].
“http://example.org/stuff/1.0/last_name": [{
”@language": "fr",
"@value": "Durand"}
...
```

(2) JSON-LD

```
...
@prefix ex1: <http://example.org/stuff/1.0/>
<http://www.univ-pau.fr> ex1:lab <http://liuppa.univ-pau.fr/live/> ;
ex:nameprof [ ex:first_name
"Sebastien"^^<http://www.w3.org/2001/XMLSchema#string>;
ex:last_name "Durand"@fr ]
```

(3) N3

```
_:N17 http://example.org/stuff/1.0/first_name
"Sebastien"^^<http://www.w3.org/2001/XMLSchema#string>
_:N17 <http://example.org/stuff/1.0/last_name> "Durand"@fr
<http://www.univ-pau.fr> <http://example.org/stuff/1.0/nameprof>
_:N17 <http://www.univ-pau.fr> <http://example.org/stuff/1.0/lab>
<http://liuppa.univ-pau.fr/live/>
```

(4) N-Triple

(b) RDF Serializations of Graph 1

Fig. 1. RDF description (graph and serialization) examples

and contain several redundancies in terms of both: the structure of the graph and/or the serialization format of the resulting RDF-based file. For instance, graphs 1 and 2 in Figure 1, which have been created by two different users, describe the same RDF information even though they have different structures (e.g., duplication of nodes and edges). Additionally, even more redundancies and disparities[1] will occur when serializing both graphs (e.g., the same RDF resource or blank node can be serialized in different ways, language and data-type declarations can be specified or omitted, namespaces can have different/duplicate prefixes, etc.). Consequently, such RDF redundancies and disparities would naturally have a negative impact on the processing of RDF databases (including storage, querying, mapping, annotating, versioning, etc.).

In this paper, we address the problem of cleaning RDF descriptions by introducing a new method as a means of transforming different RDF descriptions of the same RDF statements into one single (normalized) representation. Our method targets RDF modeling on two levels: 1) the structure/graph (logical) level, by eliminating redundancies in RDF graphs, which is typically useful in graph-based RDF querying, mapping, and versioning applications, and 2) the serialization (physical) level, by eliminating redundancies and disparities in the syntactic structure, and adapting it to the application domain, in order to optimize storage space and loading time. The rest of this article is organized as follows. Section 2 reviews background and related work in RDF normalization. Section 3 presents motivating examples, highlighting different normalization features left unaddressed by most existing approaches. Our method for normalizing RDF descriptions (graphs and serializations) is detailed in Section 4. In Section 5, we present our prototype and illustrate experimental results. We finally conclude in Section 6.

[1] We use disparities to designate different serializations of the same information.

2 Background and Related Work

2.1 Basic Notions

Definition **1** **(Statement** ***[st]*****)** *An RDF statement expresses a relationship between two resources, two blank nodes or one resource and blank node. It is defined as an atomic structure consisting of a triple with a Subject (s), a Predicate (p) and an Object (o), noted st:$< s, p, o >$, w.r.t. a specific vocabulary V (see Def. 3), where: (a) $s \in U \cup BN$ is the subject to be described (U is a set of Internationalized Resource Identifiers and BN is a set of Blank Nodes), (b) $p \in U$ refers to the properties of the subject, and (c) $o \in U \cup BN \cup L$ is the object (L is a set of Literals)* ♦

The example presented in Fig. 1.1.a underlines 4 statements with different resources, properties, and blank nodes such as st_1: <`http://www.univ-pau.fr`, $ex{:}nameProf$, $UX >$, st_2: $< UX$, $ex{:}first\_name$, “*Sebastien*”ˆˆ$xsd : string >$, st_3: $< UX$, $ex : last\_name$, “*Durand*” $>$, and st_4: <`http://www.univ-pau.fr`, *ex:lab*, `http://www.univ-pau.fr`>.

Definition **2** **(RDF Graph** ***[G]*****)** *An RDG graph is is a directed labeled graph made of a set of $< s, p, o >$ statements in which each statement is represented as a node-edge-node link* ♦

In the remainder of the paper, *”RDF graph”* and *”RDF logical representation”* are used interchangeably.

Definition **3** **(RDF Graph Vocabulary** ***[V]*****)** *An RDF Graph Vocabulary is the set of all values occurring in the RDF graph, i.e., $V = U \cup L \cup BN$* ♦

Definition **4** **(External Vocabulary** ***[QN]*****)** *An RDF External Vocabulary is a set of QNames*[1] *(QN) to represent IRI references $\{qn_1, qn_2, \ldots, qn_n\}$. Each qn_i is a tuple $< px_i, ns_i >$ where px_i is a prefix*[2] *(e.g., foaf, ex, dc,...) and ns_i is a namespace* ♦

For instance *QN*={(**ex**, `http://example.org/stuff/1.0`), (**mypx**, `http://ucsp.edu.pe`)}, where “ex” is a standard prefix, “mypx” is a local prefix, and `http://example.org/stuff/1.0` and `http://ucsp.edu.pe/` are namespaces.

Definition **5** **(RDF File** ***[F]*****)** *An RDF file is defined as an encoding of either a set of RDF statements or an RDF graph, using a predefined serialization format (e.g., RDF/XML, Turtle, N3, etc.)* ♦

In the remainder of the paper, *RDF file*, *RDF serialization* and *RDF physical representation* are used interchangeably. Also, the following functions **R**, **U**, **L**, **BN**, **ST**, **NS** and **Px** will be used respectively to return all the Resources (IRIs and literals), IRIs, Literals, Blank Nodes, statements, namespaces and prefixes of a graph *G* or a file *F*.

[1] `http://www.w3.org/TR/REC-xml-names/`

[2] Following the W3C Recommendation, we consider that all the prefixes have to be unique for each namespace

2.2 Related Work

The need for RDF normalization has been identified and discussed in various domains, ranging over knowledge representation, data integration, and service and semantic data mediation. Yet, few existing studies have addressed the issues of logical (graph) and physical (syntax) RDF normalization.

Knowledge Representation, Integration and Semantic Mediation. Various approaches have been developed to normalize knowledge representation in RDF, namely in the bioinformatics domain [14,11,6,1,10]. In [14], the authors provide an approach to map LexGrid [11], a distributed network of lexical resources for storing, representing and querying biomedical ontologies and vocabularies, to various Semantic Web (SW) standards, namely RDF. They introduce the LexRDF project which leverages LexGrid, mapping its concepts and properties to standard - normalized - RDF tagging, thus providing a unified RDF based model (using a common terminology) for both semantic and lexical information describing biomedical data. In a related study [10], the authors introduce the Bio2RDF project, aiming to create a network of coherent linked data across life sciences databases. The authors address URI normalization, as a necessary prerequisite to build an integrated bioinformatics data warehouse on the SW, where resources are assigned URIs normalized around the bio2rdf.org namespace. Several approaches have developed semantic mediators (translators), in order to convert information from one data source to another following the data format which each system understands [8,7]. Most studies consider the original data to be well organized (normalized), thus the resulting RDF data would allegedly follow. Note that in most of these projects, issues of redundancies in RDF logical and syntax representations are mostly left unaddressed.

RDF Graph (Logical) Normalization. In [5], the authors discuss some of the redundancies which can occur in a traditional RDF directed labeled graphs. Namely, an RDF graph edge label (i.e., a predicate) can occur redundantly as the subject or the object of another statement (e.g., $< dbpedia : Researcher, dbpedia : Workplace, dbpedia : University >$ and $< dbpedia : Workplace, rdf : type, dbpedia : Professional >$). Hence, the authors in [5] introduce an RDF graph model as a special bipartite graph where RDF triples are represented as ordered 3-uniform hypergraphs where edge nodes correspond to the $< subject, predicate, object >$ triplet constituents, ordered following the statement's logical triplet ordering. The new model is proven effective in reducing the predicate-node duplication redundancies identified by the authors. In subsequent studies [3,4], the authors address the problem of producing RDF normal forms and evaluating the equivalence among them. The studies in [3,4] specifically target the RDFS vocabulary with a set of reserved words to describe the relationships between resources (e.g., rdfs:type, rdfs:range, rdfs:domain, etc.). They provide a full-fledged theoretical model including notions such as: *RDF lean* (minimal) *graph* as a graph preserving all URIs of its origin graph while having fewer blank nodes, and *RDF normal form* as a minimal and unique representation of an RDF graph, among others. In [13,9], the the authors introduce an algorithm allowing to transform an RDF graph into a standard form, arranged in a deterministic way, generating a cryptographically-strong hash identifier for the graph, or digitally signing it. The algorithm takes a JSON-LD input format, and provides an output in N-triple serialization while relabeling certain nodes and erasing certain redundancies. Our approach completes the latter studies by targeting logical (graph) redundancies which were out of the scope of [3,4,5,13,9], namely distinct edge (predicate) duplication, blank node (subject/object) duplication, and combined edge and node duplication.

RDF Syntax (Physical) Normalization. At the physical (syntactic) level, the authors in [16] introduce a method to normalize the serialization of an RDF graph using XML grammar (DTD) definitions. The process consists of two steps:(a) Defining an XML grammar (DTD) to which all generated RDF/XML serializations should comply, (b) Defining SPARQL query statements to query the RDF dataset in order to return results, consisting of serializations compliant with the grammar (DTD) at hand. This is comparable to the concept of semantic mediation using SPARQL queries [8]. Note that the SPARQL statements are automatically generated based on the grammar (DTD).

To sum up, our approach completes and builds on existing methods to normalize RDF information, namely [3,4,5,13,16,9], by handling logical and physical redundancies and disparities which were (partially) unaddressed in the latter.

3 Motivating Example

We discuss here the motivations of our work, highlighting two different levels: i) logical redundancies, and ii) physical redundancies and disparities.

3.1 Logical (Graph) Redundancies

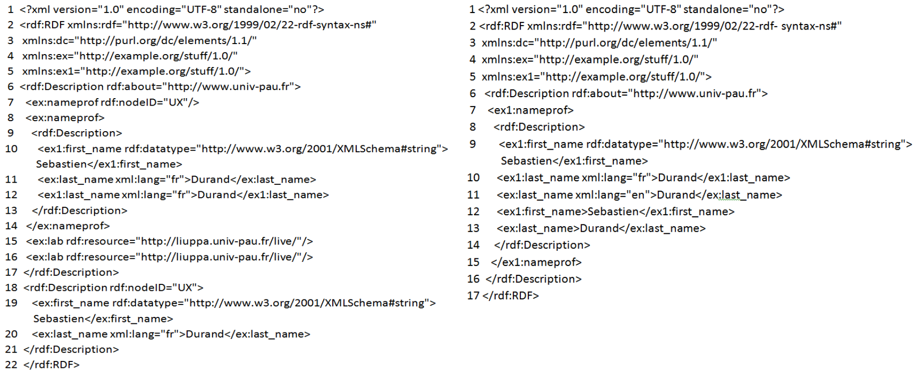
Consider the example given in Fig. 1.a.2. One can easily see several kinds of redundancies:

- **Problem 1 - Edge Duplication**: where identical edges, designating identical RDF predicates, appear more than once,
- **Problem 2 - Node duplication**: where identical nodes, designating identical RDF subjects and/or objects, appear more than once, e.g., in Fig. 1.a.2 highlighting Blank Node duplication and Literal duplication respectively.

3.2 Physical (Serialization) Disparities

Consider now Fig. 2.a which represents a possible serialization of the RDF graph in Fig. 1.a.2 encoded in the RDF/XML format. One can see that several types of redundancies and disparities are introduced: some are inherited from the logical level, while others appear at the physical (serialization) level:

- **Problem 3 - Namespace duplication**: where two different prefixes are used to designate the same namespaces (*ex* and *ex1* in lines 4-5 in Fig. 2.a),



```
1 <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3  xmlns:dc="http://purl.org/dc/elements/1.1/"
4  xmlns:ex="http://example.org/stuff/1.0/"
5  xmlns:ex1="http://example.org/stuff/1.0/">
6 <rdf:Description rdf:about="http://www.univ-pau.fr">
7   <ex:nameprof rdf:nodeID="UX"/>
8   <ex:nameprof>
9     <rdf:Description>
10      <ex1:first_name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
        Sebastien</ex1:first_name>
11      <ex:last_name xml:lang="fr">Durand</ex:last_name>
12      <ex1:last_name xml:lang="fr">Durand</ex1:last_name>
13    </rdf:Description>
14  </ex:nameprof>
15  <ex:lab rdf:resource="http://liuppa.univ-pau.fr/live/"/>
16  <ex:lab rdf:resource="http://liuppa.univ-pau.fr/live/"/>
17 </rdf:Description>
18 <rdf:Description rdf:nodeID="UX">
19    <ex:first_name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      Sebastien</ex:first_name>
20    <ex:last_name xml:lang="fr">Durand</ex:last_name>
21 </rdf:Description>
22 </rdf:RDF>
```

(a) Of the RDF graph in Fig. 1.a.2

```
1 <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf- syntax-ns#"
3  xmlns:dc="http://purl.org/dc/elements/1.1/"
4  xmlns:ex="http://example.org/stuff/1.0/"
5  xmlns:ex1="http://example.org/stuff/1.0/">
6   <rdf:Description rdf:about="http://www.univ-pau.fr">
7    <ex1:nameprof>
8      <rdf:Description>
9        <ex1:first_name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
         Sebastien</ex1:first_name>
10      <ex1:last_name xml:lang="fr">Durand</ex1:last_name>
11      <ex:last_name xml:lang="en">Durand</ex:last_name>
12      <ex1:first_name>Sebastien</ex1:first_name>
13      <ex:last_name>Durand</ex:last_name>
14     </rdf:Description>
15    </ex1:nameprof>
16  </rdf:Description>
17 </rdf:RDF>
```

(b) With syntactic disparities

Fig. 2. RDF/XML serializations

- **Problem 4 - Unused namespace**: where the namespaces are declared but never called in the body of the document (*dc* in line 3 in Fig. 2.a).
- **Problem 5 - Node order variation**: i.e., node siblings in the RDF description might be ordered differently when serialized (e.g., nodes in lines 6-16 in Fig. 2.b follow the order of appearance of XML elements, which can be re-ordered differently in another serializations).
- **Problem 6 - Handling typed elements**: objects (literals) elements can be typed or not (lines 9-12 in Fig. 2.b).
- **Problem 7 - Handling language tags**: Distinguishing between identical literals having different language tags (lines 10-11 in Fig. 2.b).

One can clearly realize the compound effect of missing logical and physical normalization when contrasting the serialization in Fig. 1.b.1 with those in Fig. 2, all of which represent the same RDF information (cf. Section 1). In the following, we develop our proposal addressing the above problems. Due to space limitations, only RDF/XML format is used in what follows to illustrate RDF serialization results.

4 RDF Normalization Proposal

We first provide a set of definitions, rules, and properties before developing our process.

4.1 Definitions

DEFINITION **6 (Extended Statement** *[$\boldsymbol{st}^+$]***)** *An extended statement is a more expressive representation of a statement* (st)*, denoted as:* $\boldsymbol{st}^+$*:*$< s', p', o' >$ *where:*

- s'*:*$< s, ts >$ *is composed of the subject value* (s) *and its type* $(ts \in \{u, bn\})$

- *p':$< p, dt, lng >$ is composed of the predicate value (p), its datatype $dt \in DT$*[1] *$\cup$ $\{\bot\}$, and language tag $lng \in Lang$*[2] *$\cup$ $\{\bot\}$. $\bot$ represents a "null" value.*
- *o':$< o, to >$ is composed of the object value (o) and its type $to \in \{u, bn, l\}$*

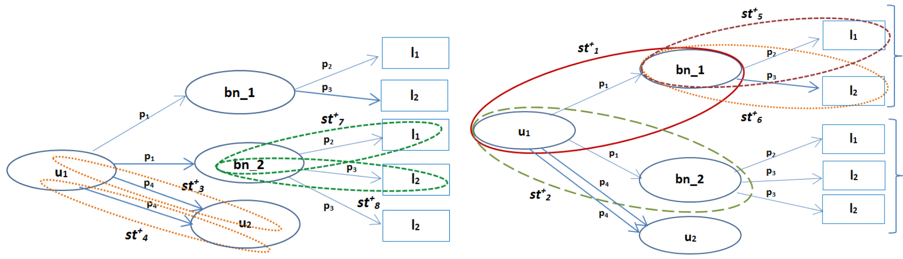
The following notation is adopted to represent an extended statement:

$$st^{+} :< s_{ts\in\{u,bn\}}, p_{dt}^{lng}, o_{to\in\{u,bn,l\}} \blacklozenge$$

Based on the example of Fig. 1.1.a, st_1 becomes st_1^+: <`http://www.univ-pau.fr`$_u$,ex:nameProf$_\bot^\bot$, UX$_{bn}$ >. The function $\boldsymbol{ST^+}$ will be used in the following to return all the statements of an RDF Description.

Definition **7** **(Extended Statement Equality** *[$=_{st}$]***)** *Two extended statements st_i^+ and st_j^+ are equal, noted $st_i^+ =_{st} st_j^+$, if and only if: 1) $st_i^+.s' = st_j^+.s'$, 2) $st_i^+.p' = st_j^+.p'$, and 3) $st_i^+.o' = st_j^+.o'$* ♦

In Fig. 3.a, $st_3^+ =_{st} st_4^+$ since they share the same subject (i.e., u_1), the same predicate (i.e., p_4), and the same object (i.e., u_2).



(a) With equal and different extended statements

(b) Of extended statement containment depicting the graph in Fig. 1.a.2

Fig. 3. RDF statement examples

Definition **8** **(Outgoings** *[**O**]***)** *Given an extended RDF statement st_i^+, the outgoings of st_i^+, noted $O(st_i^+)$, designate the set of extended statements having for subject the object element o_i' of st_i^+:*

$$O(st_i^+ :< s_i', p_i', o_i' >) = \{st_j^+ :< (o_i', p_j', o_j' >, ..., st_n^+ :< o_i', p_n', o_n' >\}\blacklozenge$$

In Fig. 3.b, we identify the following outgoings of st_1^+: $O(st_1^+) = \{st_5^+, st_6^+\}$ $= \{< bn\_1_{bn}, p2_{string}^{\bot}, l_{1l} >, < bn\_1_{bn}, p3_{\bot}^{fr}, l_{2l} >\}$

Definition **9** **(Statement Containment** *[$\preceq$]***)** *An extended statement st_i^+ is said to be contained in another extended statement st_j^+, noted $st_i^+ \preceq st_j^+$, if: 1) $O(st_i^+) \subseteq O(st_j^+)$, 2) both st_i^+ and st_j^+ have the same subject and predicate, and 3) the object type (*to*) in both statements is a blank node (*bn*)* ♦

[1] DT is a set of datatypes: string, number, date, etc.

[2] Lang is a set of language tags: @fr, @en, etc.

In Fig. 3.b[1], $st_1^+ \preceq st_2^+$ since they share the same subject (i.e., s_1) and the same predicate (i.e., p_1), and have O(st_1^+) $\subseteq$ O(st_2^+) w.r.t. their outgoings.

Definition **10 (RDF Equality** *[*$=_{RDF}$*]*) *Two RDF graphs G_i and G_j are equal if all the statements of G_i occurs in G_j and vice versa. Similarly, two RDF files F_i and F_j are equal if: 1) the corresponding graphs are equal, and 2) F_i has the same encoding format (e_i) as F_j in (e_j)* ♦

4.2 Normalization Rules

In this section, we provide a set of solutions for the problems listed in Section 3. Each solution is associated to some formal rules that allow to solve them.

Solving Logical Redundancies. Given an input RDF graph G, logical redundancies related to node duplication, edge duplication and node/edge duplications (presented in Sec. 3.1) can be eliminated from G by applying the following transformation rules (proofs are provided in [15]):

Rule 1 - Statement Equality Elimination: It is designed to eliminate edge duplications and/or node duplications using equality between statements. More precisely, $\forall st_i^+, st_j^+ \in ST^+(G) \wedge i \neq j$, if $st_i^+ =_{st} st_j^+ \implies$ *remove* st_j^+. Applying this rule on G produces an RDF Graph *G'* where *G'* $=_{RDF}$ G.
Rule 2 - Statement Containment Elimination: It is designed to eliminate edge duplications between IRIs or blank nodes in the outgoing statements, and node duplications where the objects of extended statements are blank nodes and linked to other outgoing statements). More precisely: $\forall st_i^+, st_j^+ \in ST^+(G) \wedge i \neq j$, if $st_j^+ \preceq st_i^+ \implies$ *remove* $st_j^+ \cup O(st_j^+)$. Applying Rule 2 on G produces another RDF graph *G'* where *G'* $=_{RDF}$ G.

Solving Physical Disparities. Given an input RDF file F, physical disparities related to namespace duplication, unused namespaces, and node order variation (presented in Sec. 3.2) can be eliminated from F by applying the following transformation rules:

Rule 3 - Namespace Duplication Elimination: It is designed to eliminate namespace duplications along with corresponding namespace prefixes such as: $\forall qn_i, qn_j \in QN(F) \wedge i \neq j$, if $qn_i.ns_i = qn_j.ns_j \implies$ *remove* $qn_j \wedge$ *replace* $qn_j.px_j$ by $qn_i.px_i$ in (F). Applying Rule 3 on F produces another equal RDF file since duplicated $qn_j.ns_j$ and its corresponding prefix $qn_j.px_j$ have been removed, while replacing $qn_j.px_j$ by $qn_i.px_i$ in the whole file.
Rule 4 - Unused Namespace Elimination: It is designed to eliminate the unused namespaces[2] with their respective prefixes such as: $\forall qn_i.ns_i \in NS^+(F)$,

[1] st_i^+, u_i, p_i, bn_i, and l_i represent corresponding extended statements, IRIs, predicates, blank nodes, and literals

[2] An unused namespace is a namespace which is mention in the serialization file but which is not use in any of the statements, i.e., it will not appear in the Graph.

if $qn_i.ns_i \notin NS^+(G) \Longrightarrow$ *remove* qn_i. Applying Rule 4 on F produces another equal file where unused $qn_i.ns_i$ and respective prefix px_i have been removed.
Rule 5 - Reordering: It is designed to solve the varying node order problem by imposing a predefined (user-chosen) order on all statements of an RDF File $\tilde{F}$. More formally: $\forall st_i^+, st_j^+ \in ST^+(F) \Longrightarrow st_i^+ <_{\tilde{p}} st_j^+$ where $\tilde{p} :< iorder, sortc >$ is a tuple composed of an indexing order "*iorder*" and a sorting criterion "*sortc*". For *iorder*, we follow the six indexing schemes presented in [17] (SPO, SOP, PSO, POS, OSP, OPS) since it is the combination of the three elements of the statement (subject, predicate, object) in an RDF Description. For *sortc*, we adopt *asc* and *des* to represent asending and descending order respectively. The default value for the parameter $\tilde{p}$ is $< sop, asc >$ representing an ascending order of statements w.r.t. their subjects / objects / predicates (sop). Applying Rule 5 on F produces an equal RDF file where all the statements have been ordered following the (user-chosen) order type parameter $\tilde{p}$.

4.3 Normalization Properties

Our approach verifies the following properties characterizing the quality of the normalization process. This corresponds to the notion of Information Reusability discussed in existing studies[1]. Proofs are provided in [15].

Property **1 (Completeness)** *An RDF Description D_i (graph or file) is said to be complete regarding D_j if D_i preserves and does not lose any information in D_j, i.e., each resource, statement and namespace of D_j has a corresponding resource, statement and namespace in D_i* •

Property **2 (Minimality)** *An RDF Description D_i is said to be minimal if all the resources, statements and namespaces of D_i are unique (i.e., they do not have duplicates in D_i) and all the namespaces are used* •

Property **3 (Compliance)** *An RDF File F_i is said to be compliant if it verifies all the rules of the W3C standard in producing a valid file based on an RDF Graph G_i, i.e., its structure remains compliant with RDF serialization standards (e.g., RDF/XML)* •

Property **4 (Consistency)** *An RDF Description D_i is said to be consistent if D_i verifies the completeness, minimality and compliance properties to ensure the data quality of the description* •

4.4 Normalization Process

The inputs of our RDF normalization process are: a) the RDF graph (logical representation) or RDF file (physical representation) to be normalized, and b) user parameters related to the RDF output form and prefix renaming, enabling the user to tune the results according to her requirements (see Fig. 4). In the following, we detail each of them.

[1] This is comparable to the notion of *map function* in [4] except that the authors do not consider namespaces.

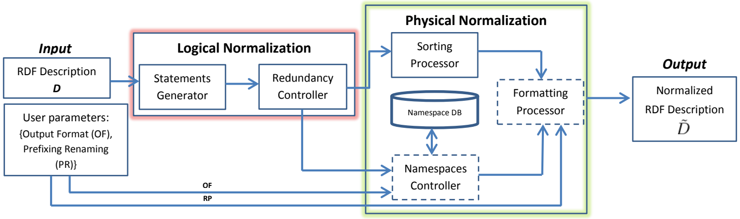


Fig. 4. Architecture of our RDF normalization process

Logical Normalization. Algorithm 1 provides the pseudo-code to remove redundancies from an RDF graph. It starts by erasing the statement redundancies having IRIs or literals. Consequently, it removes the equal statements as well as contained ones with duplicated blank nodes (bn) (and all the outgoings O derived of the bn) following Def. 9.

Algorithm 1 Removing Redundancy

```
Input: $st^+[]$ //List of Extended Statements of the RDF Description
Output: $st^+[]$ //List of Extended Statements without duplication
N=$st^+$.length(); //Number of Statements in the list
for i=1, i ≤ N, i++ do
    for j=i+1, j ≤ N, j++ do
        if $st^+$[i].to = "IRI" or $st^+$[i].to = "Literal" and $st^+[\text{i}] =_{st} st^+[\text{j}]$ then
            Remove $st^+[j]$; // remove statement duplication following Rule 1
        else
            if $st^+[j]$.to = "bn" and $st^+[i]$.to = "bn" and $st^+[i]$.s = $st^+[j]$.s and $st^+[i]$.p = $st^+[j]$.p
            and ($st^+[i] \preceq st^+[j]$ or $st^+[j] \preceq st^+[i]$) then
                Remove $st^+[j]$ and all its outgoings // remove blank node duplication - Rule 2
            else
                if $st^+[i]$.o = $st^+[j]$.o then
                    Remove $st^+[j]$ // remove statement duplication following Rule 1
```

Physical Normalization. It is divided into three components based on the types of physical disparities being processed:

1. ***Namespaces Controller (NC):*** it controls namespace duplication by eliminating the redundancies (Rule 3) and the unused namespaces (Rule 4) in the RDF File. The input of this component is the prefix renaming parameter, which allows to customize the renaming of the prefixes while providing a unique way to normalize them.
2. ***Sorting Process (SP):*** Rule 5 establishes the node order variation, to have a unique specification of the statements in the output serialization. It considers the parameter $\tilde{p} :< iorder, sortc >$ given by the user according to the targeted applications.
3. ***Formatting Process (FP):*** it allows to: a) choose a specific form for the output RDF file, b) manage the variety of blank node serializations, and c) manage the datatypes and the languages. Our current solution allows three different output forms (other forms could be devised based on user/application requirements):

   Flat: it develops each RDF description one by one as a single declaration, i.e., each subject has one declaration in the file.

Compact: it nests the RDF description, i.e., each description may have another description nested in its declaration.
Full compact: dedicated to RDF/XML format, it nests the RDF description, uses the ENTITY XML construct to reduce space by providing an abbreviation for IRIs[1], reuses the variables in the RDF file, and uses attributes instead of properties for the blank node serialization.

Providing different output types is necessary to satisfy the requirements of different kinds of RDF-based applications (e.g., compact representations are usually of interest to human users when running RDF queries [4], yet less compact/more structured representations could be useful in automated processing, such as in RDF annotation recommendation [12]).

5 Experimental Evaluation

5.1 Experimental Environment

An online prototype system: ***RDF2NormRDF***[2], implemented using PHP and Java, was developed to test, evaluate and validate our RDF Normalization method. It was used to perform a large battery of experiments to evaluate: i) the effectiveness (in performing normalization), and ii) the performance (execution time) and storage space of our approach in comparison with its most recent alternatives. To do so, we considered three large datasets: i) Real_DS made of real RDF files from LinkedGeoData, ii) Syn_DS1 made of synthetic RDF files generated from the real dataset by including additional random logical redundancies and physical disparities, and iii) Syn_DS2 which consists of a variation of Syn_DS1 files including more heterogeneity (more duplications and disparities) than Syn_SD1 (cf. Table 1).

Table 1. Features of files in each Dataset

	Features	**Size**	**IRIs**	**BNs**	**Lit.**	**St**	BN	Lit.	St	**Log.**	Ns	Non-used	**Phys.**
Datasets		**kb**					Dup.	Dup.	Dup.	**Red. %**	Dup.	Ns	**Dis. %**
Real_DS	**Max**	7.2	9	0	40	77	0	34	63	77	7	8	55
	Min	2.8	9	0	7	25	0	1	11	29	1	2	40
	Avg	4.5	9	0	20	45	0	14	30	**57**	3	5	**48**
Syn_DS1	**Max**	4.3	17	12	26	63	3	7	18	69	4	2	78
	Min	0.5	2	1	3	5	0	0	0	0	0	0	0
	Avg	2.2	6	6	14	30	2	5	12	**32**	2	1	**60**
Syn_DS2	**Max**	48.6	47	129	320	784	123	307	753	98	126	3	98
	Min	0.5	2	1	3	5	0	0	0	0	0	0	0
	Avg	7.9	6	19	50	116	16	43	98	**64**	14	2	**70**

Experiments were undertaken on an Intel®Core(TM) i7 - 2600 + 3.4GHz with 8.00GB RAM, running a MS Windows 7 Professional OS and using a Sun JDK 1.7 programming environment.

5.2 Experimental Results

We evaluated the effectiveness and performance of our method, called in this section **R2NR**, in comparison with alternative methods, namely the JSON-LD normalization approach [13,9] and the HDT technique [2].

[1] `http://www.w3.org/TR/xml-entity-names/`

[2] Available at `http://rdfn.sigappfr.org/`

1. ***Effectiveness***: Results in Table 2 show that our method produces normalized RDF files that fulfill all our normalization properties and goals, in comparison with JSON-LD and HDT which miss certain logical and physical redundancies/disparities. On one hand, the ***JSON-LD method*** preserves some of the redundancies, i.e., we show that JSON-LD removes only 5% over a 32% average of logical redundancies in *Syn_DS1* and 12% over a 64% average of logical redundancies in *Syn_DS2*. However, it removes all logical redundancies from the Real_DS (all 57% of logical redundancies and 48% of physical disparities were removed). A careful inspection of JSON-LD shows that it preserves blank node duplication and certain literal duplications, which explains the results obtained with the Real_DS (since it does not contain any blank nodes). As a result, the JSON-LD approach does not satisfy the *minimality* and *consistency* properties. On the other hand, the ***HDT method*** results show, at first glance, that it successfully eliminates all logical redundancies and physical disparities. Nonetheless, a closer look at the results revealed that the HDT technique actually preserves blank node redundancies by assigning them different identifiers and/or representing them as IRIs. Hence, the HDT method actually keeps the logical redundancies (and corresponding physical disparities) and does not consequently satisfy the *completeness*, *minimality* and *consistency* properties.

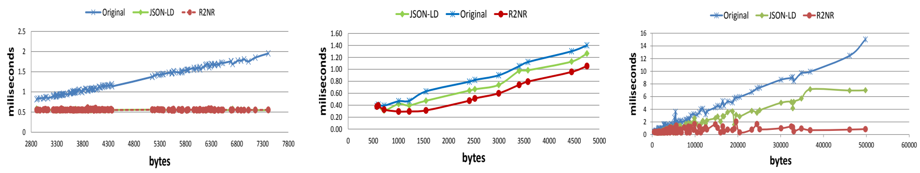
Table 2. Goals/properties achieved in Datasets after normalization processes

Goals/Properties	**Real_DS**			**Syn_DS1**			**Syn_DS2**		
	JSON-LD	**HDT**	**R2NR**	**JSON-LD**	**HDT**	**R2NR**	**JSON-LD**	**HDT**	**R2NR**
Solving log. redundancies	57%	57%	57%	5%	32%	32%	12%	64%	64%
Solving phys. disparities	48%	48%	48%	60%	60%	60%	70%	70%	70%
Preserving completeness	True	True	True	True	False	True	True	False	True
Preserving minimality	True	True	True	False	False	True	False	False	True
Preserving compliance	True	True	True	True	True	True	True	True	True
Preserving consistency	True	True	True	False	False	True	False	False	True

2. ***Performance***: In addition, we verified our approach' performance in terms of loading time and storage space, in comparison with JSON-LD and HDT.

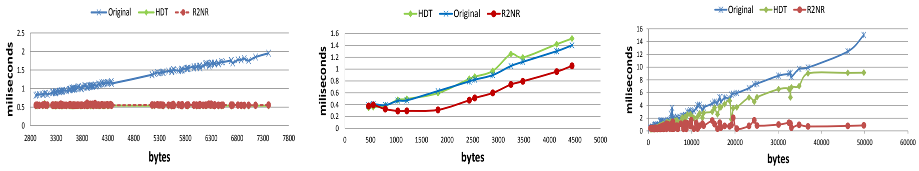
   (a) Jena loading time: The complexity of our method comes down to worst case $O(N^2)$ time where N represents the number of RDF statements in the document (cf. [15]). Jena loading time results depicted in figures 5 and 6 confirm our approach' polynomial (almost linear) time w.r.t. document size. Also, results in figures 5.a and 5.b demonstrate that our method executes faster than JSON-LD's. Note that redundancy reduction using of *JSON-LD* amounts to 5% on average file size in Syn_DS1, while our method reaches an average 27% size reduction ratio, which explains the reduction in loading time. In addition, results in Figures 6.a and 6.b show that our method remains faster than the HDT method. In fact, as shown in Table 2, the datasets generated by HDT do not have redundancies and disparities, yet contain a larger number of IRIs with no (zero) BNs. This confirms that HDT is transforming BNs into IRIs, which shows that RDF compression does not always guarantee normalization. Note that we are currently investigating this issue in more detail in a dedicated experimental study.

   (b) Storage: Neither JSON-LD nor HDT methods provide parameters to customize output format requirements as we do. They work with their prede-



(a) In Real_DS (b) In Syn_DS1 (c) In Syn_DS2

Fig. 5. Average Jena Loading Time comparison with JSON-LD method



(a) In Real_DS (b) In Syn_DS1 (c) In Syn_DS2

Fig. 6. Average Jena Loading Time comparison with HDT method

fined outputs, i.e., the JSON-LD method with N-triples, and HDT method with Bitmap Triples (BT), in comparison with our method which handles the standard RDF/XML format and thus allows developing different outputs w.r.t the target application. Figure 7 shows that our normalization method improves (reduces) the size of the RDF files in all formats of the datasets processed by JSON-LD and HDT methods.

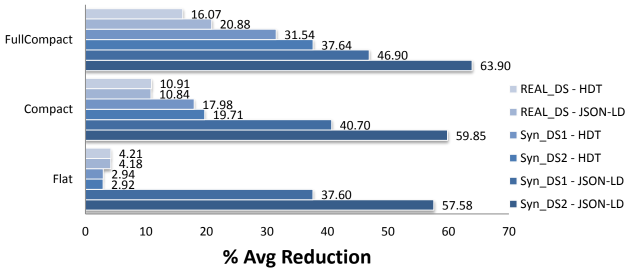


Fig. 7. Average Reduction in datasets Syn_DS1 and Syn_DS2 w.r.t. the output format

6 Conclusion

We proposed here an RDF normalization method *RDF2NormRDF* able to: i) preserve all the information in RDF descriptions, ii) eliminate all the logical redundancies and physical disparities in the output RDF description, iii) establish a unique specification of the statements in the RDF output description, and iv) consider user parameters to handle the application requirements. To our knowledge, this is the first attempt to study and integrate RDF normalization in two aspects: logical redundancies and physical disparities. Understanding that the presence of logical redundancies in the RDF descriptions causes a heavy impact in the storage and processing of information, our theoretical proposal and experimental evaluation showed that our approach yields improved normalization results with respect to existing alternatives. Ongoing works include exploiting semantic normalization to improve, not only the structure of RDF descriptions, but also their information content (as IRIs, blank nodes, and literals). Future directions also include the semantic disambiguation of RDF literals and IRIs.

Acknowledgments. This work has been partly supported by FINCyT (Fund for Innovation, Science and Technology) of Peru.

References

1. F. Belleau and et al. Bio2rdf: Towards a mashup to build bioinformatics knowledge systems. *J. Biomedical Informatics*, 41(5):706–716, 2008.
2. J. D. Fernández and et al. Binary rdf representation for publication and exchange (HDT). *J. Web Semantics*, 19:22–41, 2013.
3. C. Gutierrez and et al. Foundations of semantic web databases. In *PODS 2004*, pages 95–106. ACM, 2004.
4. C. Gutierrez and et al. Foundations of semantic web databases. *J. Comput. Syst. Sci.*, 77(3):520–541, May 2011.
5. J. Hayes and C. Gutierrez. Bipartite graphs as intermediate model for RDF. In *Int. Sym. Semantic Web*, pages 47–61. Springer, 2004.
6. G. Jiang and et al. Using semantic web technology to support ICD-11 textual definitions authoring. *J. Biomedical Semantics*, 4:11, 2013.
7. A. Kerzazi and et al. A model-based mediator system for biological data integration. *Journes Scientifiques en Bio-Informatique*, pages 70–77, 2007.
8. A. Kerzazi and et al. A semantic mediation architecture for RDF data integration. *SWAP*, pp. 3, 2008.
9. D. Longley. RDF dataset normalization. `http://json-ld.org/spec/latest/rdf-dataset-normalization/`, 2015.
10. M.-A. Nolin and et al. Building an hiv data mashup using Bio2RDF. *Briefings in bioinformatics*, 13(1):98–106, 2012.
11. J. Pathak and et al. Lexgrid: a framework for representing, storing, and querying biomedical terminologies from simple to sublime. *J. of the American Medical Informatics Assoc.*, 16(3):305–315, 2009.
12. K. Salameh, J. Tekli, and R. Chbeir. SVG-to-RDF image semantization. In *Similarity Search and Applications*, pages 214–228. Springer, 2014.
13. M. Sporny and D. Longley. RDF graph normalization. `http://json-ld.org/spec/ED/rdf-graph-normalization/20111016/`, 2013.
14. C. Tao and et al. A RDF-base normalized model for biomedical lexical grid. In *the 8th Int. Semantic Web Conference*, Series A RDF-base Normalized Model for Biomedical Lexical Grid, pp. 2, 2009.
15. Ticona-Herrera and el al. Rdf similarity. Technical report. `http://rdfn.sigappfr.org/RDFN-TR-15.pdf`, 2015.
16. D. Vrandecic and et al. RDF syntax normalization using XML validation. *Proc. of the SemRUs*, pp. 11, 2009.
17. C. Weiss, P. Karras, and A. Bernstein. Hexastore: sextuple indexing for semantic web data management. *Proc. VLDB Endow.*, 1(1):1008–1019, 2008.