Binary RDF Representation for Publication and Exchange (HDT)

Javier D. Fernández[a,*], Miguel A. Martínez-Prieto[a,b], Claudio Gutiérrez[b], Axel Polleres[c,d], Mario Arias[a,c]

[a]*DataWeb Research, Department of Computer Science, University of Valladolid, E.T.S. de Ingeniería Informática, Campus Miguel Delibes, 47011 Valladolid, Spain.*
[b]*Department of Computer Science, University of Chile, Avenida Blanco Encalada 2120, 837-0459 Santiago, Chile.*
[c]*Digital Enterprise Research Institute, National University of Ireland, Galway, IDA Business Park, Lower Dangan, Galway, Ireland.*
[d]*Siemens AG Österreich, Siemensstrasse 90, 1210 Vienna, Austria.*

Abstract

The current *Web of Data* is producing increasingly large RDF datasets. Massive publication efforts of RDF data driven by initiatives like the Linked Open Data movement, and the need to exchange large datasets has unveiled the drawbacks of traditional RDF representations, inspired and designed by a document-centric and human-readable Web. Among the main problems are high levels of verbosity/redundancy and weak machine-processable capabilities in the description of these datasets. This scenario calls for efficient formats for publication and exchange.

This article presents a binary RDF representation addressing these issues. Based on a set of metrics that characterizes the skewed structure of real-world RDF data, we develop a proposal of an RDF representation that modularly partitions and efficiently represents three components of RDF datasets: Header information, a Dictionary, and the actual Triples structure (thus called HDT). Our experimental evaluation shows that datasets in HDT format can be compacted by more than fifteen times as compared to current naive representations, improving both parsing and processing while keeping a consistent publication scheme. Specific compression techniques over HDT further improve these compression rates and prove to outperform existing compression solutions for efficient RDF exchange.

Keywords: RDF, Binary formats, Data compaction and compression, RDF metrics.

1. Introduction and Motivation

The original RDF (*Resource Description Framework*) W3C Recommendation [38] (from 1999) defines RDF as a foundation for processing metadata and establishes its broad goal as a mechanism for describing resources. This conception was clearly influenced by a *document-centric* perspective of the Web as it is stated through the examples of RDF applications, such as the description and annotation of web page collections that represent a single logical document or the intellectual property rights of web pages. Although the current Recommendation [8] (from 2004) shares this original perspective, it also devises an evolution by suggesting the use of RDF "*to do for machine processable information (application data) what the WWW has done for hypertext: to allow data to be processed outside the particular environment in which it was created, in a fashion that can work at Internet scale*". This latter perspective along with increasing adoption driven by efforts such as the Linked Data[1] initiative has made RDF evolve from a simple format for metadata to a universal exchange format.

The mainstream RDF serialization syntaxes share the scope of the original perspective. The intended goal of the original RDF/XML representation design was to add small descriptions (metadata) to documents, to protocols, to annotate web pages or to describe services. Likewise, representations like N3 [10], Turtle [9] and RDF/JSON [2], and more recently RDFa[2],although having reduced the verbosity of the original format, are still dominated by this document-centric view. Today, as one of the major trends in the development of the Web is RDF publication and exchange at large scale, *i.e.*, making RDF data (publicly) available for diverse purposes and users, the need to consider RDF under a data-centric

*Corresponding author
Email addresses: jfergar83@gmail.com (Javier D. Fernández), migumar2@infor.uva.es (Miguel A. Martínez-Prieto), cgutierr@dcc.uchile.cl (Claudio Gutiérrez), axel.polleres@siemens.com (Axel Polleres), mario.arias@deri.org (Mario Arias)

[1]http://linkeddata.org
[2]http://www.w3.org/TR/2012/REC-rdfa-core-20120607/

view is becoming indispensable.

The emerging Web of Data comprises a variety of very large datasets from diverse fields such as bioinformatics, social networks or structured knowledge extracted from Wikipedia. The Linked Data initiative promotes the use of standards (such as RDF and HTTP) to publish such structured data on the Web and to connect it by reusing dereferenceable identifiers between different data sources [13]. The latest studies of the so-called Linked Open Data (LOD) cloud[3] estimate that more than 31 billion RDF triples are being shared and increasingly linked (close to half a billion links), which results in one huge interconnected RDF graph, organized in datasets from different providers. Hence, when consuming (parts of) this huge graph a major problem is to manage, exchange, and consume these datasets efficiently. Similar problems arise when managing this information in mobile devices; together with memory constrains, these devices can face additional transmission costs [39].

An analysis of published RDF datasets reveals several insufficiencies of existing RDF serializations for publishing and exchanging RDF at large scale. Firstly, even though there are some approaches (*e.g.* VoiD [4]) to add provenance and other metadata (such as statistics or a content summary) when publishing RDF datasets, such information – which can be useful to guide consumers – is usually neither complete nor systematically published along with the dataset. In many cases this additional information is given in non machine-readable formats (*e.g.* natural language) in the web page of the dataset, making it difficult to relate to the actual dataset.

Moreover, basic data operations (such as simple lookup) have to deal with the sequentiality of the information in files, requiring to parse the whole data. Publishing, exchanging, and consuming large RDF datasets is not supported in a standardized fashion. This state of affairs does not scale to a Web where very large datasets will soon be produced dynamically and automatically. Furthermore, most data have to be *machine-understandable* in line with the aim of the original Semantic Web project.

From the above, we can conclude that the process of publishing and exchanging large RDF datasets should comply with some basic requirements. At the *logical level*, large-scale datasets should have standard metadata, like provenance (source, providers, publication date, etc.), editorial metadata (publisher, date, version, etc.), dataset statistics (size, quality, type of data, basic parameters of the data) and intellectual property information (types of copy[left—right]s). At the *physical level*, RDF representation at large scale should permit efficient processing, management and exchange (between systems and memory-disk movements), thus minimizing redundancy, while at the same time guaranteeing modularity. At the *operational level*, desirable features include native support for simple query patterns. The state-of-the-art on publishing and exchanging large datasets (*e.g.* from an RDF data store) would typically be to first dump the data into one file using one of the existing RDF serialization formats, and then, due to the large size of the data, possibly compress this serialization with a generic compression algorithm. However, there is no agreed way to publish such a dump along with additional metadata and it is hardly usable natively, *i.e.*, without an expensive processing using an appropriate external tool (an RDF store, a visualizer, etc.).

The present work provides a novel RDF publication and serialization format which addresses the above-mentioned challenges. First of all, one needs to understand the structure of real-world huge RDF graphs, which will guide the design. To this end, we propose a set of specific metrics for RDF datasets which reveal the underlying structure and composition of the graph. Then, we introduce the new representation format (*Header-Dictionary-Triples*: `HDT`) that modularizes the data and uses the skewed structure of big RDF graphs [25, 46, 53] to achieve large spatial savings. `HDT`, following the requirements delineated above, is based on three main components:

- A *header*, including metadata describing the RDF dataset. It serves as an entry point to the information on the dataset.

- A *dictionary*, organizing all the identifiers in the RDF graph. It provides a catalog of the RDF terms (URIs, blank nodes, literals) mentioned in the graph with high levels of compression.

- A *triples* component, which comprises the pure structure of the underlying RDF graph, *i.e.* compactly encodes the set of triples while avoiding the noise produced by long labels and repetitions.

We make use of succinct data structures and simple compression notions to get a practical implementation for `HDT`. Our design, besides gaining modularity and compactness, also addresses other important features: 1) it allows indexed access to the RDF graph, and 2) it uses a specific technique for RDF compression (referred to as `HDT-Compress`) showing a technique able to outperform universal compression algorithms.

Figure 1 shows a step-by-step description of the process to obtain the `HDT` representation of an RDF graph.

[3]http://www4.wiwiss.fu-berlin.de/lodcloud/state/

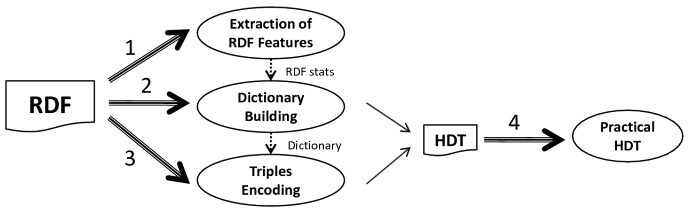


Figure 1: A Step-by-step construction of HDT from a set of triples. The last step covers practical decisions to get a concrete implementation of HDT.

The first three steps extract basic RDF features necessary to build the dictionary and the underlying graph, as well as information that will be included in the header. The fourth step covers some practical decisions in order to get a concrete implementation for HDT.

When it comes to publish or exchange large RDF datasets, the advantages of HDT compared to existing RDF serialization formats can be summarized as follows. (i) HDT is more compact, thus – depending on the concrete application – saving storage space, communication bandwidth and data transfer time. (ii) HDT is modular, cleanly separating the dictionary from triples (the graph structure), including a standard header with metadata about the dataset. (iii) HDT permits basic data operations allowing access to parts of the graph without the need to process/parse it in its entirety.

The paper is organized as follows[4]. Section 2 defines a set of metrics to characterize the structural RDF features. Section 3 presents the HDT format by an individual description of each component: Header, Dictionary, and Triples. Section 4 details the practical implementation approach for HDT. Section 5 characterizes HDT compacting ability. We perform an empirical study which analyzes the current HDT features on real-world datasets. Section 6 reviews the related work and the state-of-the-art of RDF structural studies, representations and applications. Section 7 gives conclusions and addresses future work. Appendix Appendix A provides an empirical study on characterizing RDF through the metrics presented in Section 2. Finally, a concrete HDT syntax specification is provided in Appendix Appendix B.

2. Characterizing RDF

Despite RDF is being widely used, its structural properties in real-world deployments are still neither well-known nor exploited. Several studies confirm the presence of power-law distributions, in term of frequencies [25], resources [46] and schemas [53], while others give some indications about RDF compression potential [27]. Hogan et al.'s work [32, 31] confirms many of those observations and additionally analyzes popularity in terms of interlinkage and publishing quality of RDF online, particularly focusing on compliance with Linked Data principles. Among statistical analysis, a relevant related work [33, 34] defines some equivalent metrics to our proposal below, such as cardinalities for (subject,predicate) and (predicate,object) pairs.

The objective of this section is to present a theoretical and empirical study on real-world RDF structure and properties, in order to determine common features and characterize real-world RDF data. This can lead to better dataset designs, efficient RDF data structures, indexes and compression techniques.

2.1. Metrics for RDF graphs

RDF is typically formalized as follows. Assume infinite, mutually disjoint sets U (RDF URI references), B (Blank nodes), and L (RDF literals). A triple $(s, p, o) \in (U \cup B) \times U \times (U \cup B \cup L)$ is called an RDF triple [29], in which s is the subject, p the predicate and o the object. Note that (s, p, o) can be represented as a direct edge-labeled graph $s \xrightarrow{p} o$.

We note that RDF interpretation as a graph can be misleading. RDF can be represented as an edge-labeled graph for visualization but, in fact, it can not be considered a graph in the standard sense because the predicates can again appear as nodes of other edges [30]. Thus, the application of well-established methods from graph theory presents problems. For instance, traditional graph metrics must be reconsidered as well.

We provide some specific parameters to characterize RDF data, and show that they have skewed power-law distributions, particularly remarkable on subjects and objects, establishing the basis of our representation. For our purposes, few indicators of the graph structure are sufficient. We follow [47, 29] for graph notation, with

[4]A preliminary version of this article appeared in *Proc. 9th International Semantic Web Conference (ISWC)*, pp. 193–208, 2010. It has also led to a W3C Member Submission (30 March 2011), http://www.w3.org/Submission/2011/SUBM-HDT-20110330/.

no distinction between URIs, Blank nodes and Literals[5].

Let G be an RDF graph, and S_G, P_G, O_G be the sets of subjects, predicates and objects in G.

Definition 1 (subject out-degrees)**.** *Let G be an RDF graph, and let $s \in S_G$ and $p \in P_G$.*

1. *The* out-degree *of s, denoted $deg^-(s)$, is defined as the number of triples in G in which s occurs as subject. Formally, $deg^-(s) = |\{(s,y,z) \mid (s,y,z) \in G\}|$. The* maximum out-degree, *$deg^-(G) = max_{s \in S_G}(deg^-(s))$, and the* mean out-degree, *$\overline{deg^-}(G) = \frac{1}{|S_G|}\Sigma_{s \in S_G} deg^-(s)$, are defined as the maximum and mean out-degrees of all subjects in S_G.*

2. *The* partial out-degree *of s with respect to p, denoted $deg^{--}(s,p)$, is defined as the number of triples of G in which s occurs as subject and p as predicate. Formally, $deg^{--}(s,p) = |\{(s,p,z) \mid (s,p,z) \in G\}|$. The* maximum partial out-degree *of G, $deg^{--}(G) = max_{(s,p) \in S_G \times P_G}(deg^{--}(s,p))$, and* mean partial out-degree, *$\overline{deg^{--}}(G) = \frac{1}{|S_G \times P_G|}\Sigma_{(s,p) \in S_G \times P_G} deg^{--}(s,p)$, are defined as the maximum (resp. the mean) partial out-degrees of all pairs of subject-predicates of G.*

3. *The* labeled out-degree *of s, denoted $degL^-(s) = max_{s \in S_G}(degL^-(s))$, is defined as the number of different predicates (labels) of G with which s is related as a subject in a triple of G. Formally, $degL^-(s) = |\{p \mid \exists z \in O_G, (s,p,z) \in G\}|$. The* maximum labeled out-degree *of G, $degL^-(G) = max_{s \in S_G}(degL^-(s))$, and* mean labeled out-degree, *$\overline{degL^-}(G) = \frac{1}{|S_G|}\Sigma_{s \in S_G} degL^-(s)$, are defined as the maximum (resp. the mean) labeled out-degrees of all subjects of G.*

4. *The* direct out-degree *of s, denoted $degD^-(s)$, is defined as the number of different objects of G with which s is related as a subject in a triple of G. Formally, $degD^-(s) = |\{o \mid \exists y \in P_G, (s,y,o) \in G\}|$. The* maximum direct out-degree *of G, $degD^-(G) = max_{s \in S_G}(degD^-(s))$, and* mean direct out-degree, *$\overline{degD^-}(G) = \frac{1}{|S_G|}\Sigma_{s \in S_G} degD^-(s)$, are defined as the maximum (resp. the mean) direct out-degrees of all subjects of G.*

[5]Naming of blank nodes can matter in some treatments, *i.e.*, our serialization is not *canonical*. Canonical representations of RDF are, due to the structure of blank nodes, tricky to achieve in general [17].

Symmetrically, we define for objects the *in-degree*, denoted $deg^+(o)$, partial in-degree, $deg^{++}(o,p)$, labeled in-degree, $degL^+(o)$ and direct in-degree, $degD^+(o)$. Their corresponding maximums and means are denoted as $deg^+(G)$, $deg^{++}(G)$, $degL^+(G)$, $degD^+(G)$, $\overline{deg^+}(G)$, $\overline{deg^{++}}(G)$ and $\overline{degL^+}(G)$, $\overline{degD^+}(G)$.

Note that *cardinality, average cardinality, inverse cardinality* and *average inverse cardinality* in [33, 34] are equivalent to partial out-degree, average partial out-degree, partial in-degree and average partial in-degree.

Definition 2 (predicate degrees)**.** *Let G be an RDF graph, and let $s \in S_G$, $p \in P_G$ and $o \in O_G$.*

1. *The* predicate degree *of p, denoted $deg_P(p)$, is defined as the number of triples of G in which p occurs as predicate. Formally, $deg_P(p) = |\{(x,p,z) \mid (x,p,z) \in G\}|$. The* maximum predicate degree, *$deg_P(G) = max_{p \in P_G}(deg_P(p))$, and the* mean predicate degree, *$\overline{deg_P}(G) = \frac{1}{|P_G|}\Sigma_{p \in P_G} deg_P(p)$, are defined as the maximum and mean predicate degrees of all predicates in P_G.*

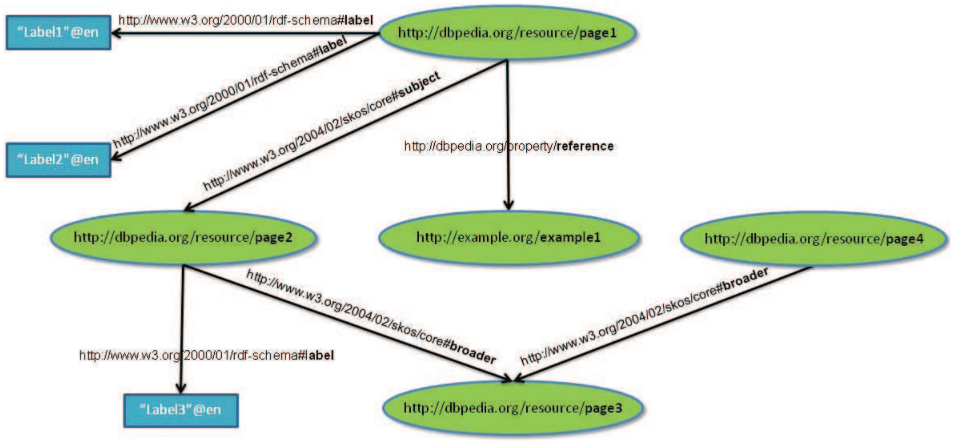
2. *The* predicate in-degree *of p, denoted $deg_P^-(p)$, is defined as the number of different subjects of G with which p is related as a predicate in a triple of G. Formally, $deg_P^-(p) = |\{s \mid \exists z \in O_G, (s,p,z) \in G\}|$. The* maximum predicate in-degree, *$deg_P^-(G) = max_{p \in P_G}(deg_P^-(p))$, and the* mean predicate degree, *$\overline{deg_P^-}(G) = \frac{1}{|P_G|}\Sigma_{p \in P_G} deg_P^-(p)$, are defined as the maximum and mean predicate in-degrees of all predicates in P_G.*

3. *The* predicate out-degree *of p, denoted $deg_P^+(p)$, is defined as the number of different objects of G with which p is related as a predicate in a triple of G. Formally, $deg_P^+(p) = |\{o \mid \exists x \in S_G, (x,p,o) \in G\}|$. The* maximum predicate out-degree, *$deg_P^+(G) = max_{p \in P_G}(deg_P^+(p))$, and the* mean predicate out-degree, *$\overline{deg_P^+}(G) = \frac{1}{|P_G|}\Sigma_{p \in P_G} deg_P^+(p)$, are defined as the maximum and mean predicate out-degrees of all predicates in P_G.*

Additionally, we will use the following property to describe RDF graph characteristics:

Definition 3 (subject-object ratio α_{s-o})**.** *The subject-object ratio $\alpha_{s-o}(G)$ of a graph is defined as the ratio of common subjects and objects in the graph G. Formally, $\alpha_{s-o}(G) = \frac{|S_G \cap O_G|}{|S_G \cup O_G|}$.*

Analogously, we define subject-predicate ratio, denoted $\alpha_{s-p}(G)$, and predicate-object ratio, $\alpha_{p-o}(G)$.


"Label1"@en
http://www.w3.org/2000/01/rdf-schema#label
http://dbpedia.org/resource/page1
http://www.w3.org/2000/01/rdf-schema#label
"Label2"@en
http://www.w3.org/2004/02/skos/core#subject
http://dbpedia.org/property/reference
http://dbpedia.org/resource/page2
http://example.org/example1
http://dbpedia.org/resource/page4
http://www.w3.org/2004/02/skos/core#broader
http://www.w3.org/2004/02/skos/core#broader
http://www.w3.org/2000/01/rdf-schema#label
"Label3"@en
http://dbpedia.org/resource/page3

Out-degrees		In-degrees		Predicate degrees		Ratios	
$deg^{-}(G)$	4	$deg^{+}(G)$	2	$degP(G)$	3	$\alpha_{s-o}(G)$	0.13
$deg^{--}(G)$	2	$deg^{++}(G)$	2	$degP^{-}(G)$	2	$\alpha_{s-p}(G)$	0
$degL^{-}(G)$	3	$degL^{+}(G)$	1	$degP^{+}(G)$	3	$\alpha_{p-o}(G)$	0
$degD^{-}(G)$	2	$degD^{+}(G)$	2				
$\overline{deg^{-}(G)}$	2.33	$\overline{deg^{+}(G)}$	1.17	$\overline{degP(G)}$	1.75		
$\overline{deg^{--}(G)}$	1.17	$\overline{deg^{++}}$	1.17	$\overline{degP^{-}(G)}$	1.50		
$\overline{degL^{--}(G)}$	2.00	$\overline{degL^{++}}$	1.00	$\overline{degP^{+}}$	1.50		
$\overline{degD^{-}(G)}$	1.17	$\overline{degD^{+}}$	1.67				

Figure 2: Various metrics for describing the structure of RDF data are shown over a small RDF graph example.

Figure 2 illustrates these properties in a small example graph inspired by DBPedia[6]. Resources are described with "labels", they can "reference" other external pages and they are categorized (using a "subject" predicate) through other DBPedia category pages, organized in hierarchies ("broader" categories). The subject out-degree indicates the cardinality of a subject node. A node with high out-degree, also called star-shaped node, will sometimes have hundreds, or even thousands, of edges (labeled edges in RDF). In conjunction with maximum and mean values, this constitutes a good evidence of these types of nodes in a given graph. Similar reasoning can be made for object in-degree, where the node is not a source, but is a common destination object node.

Partial and labeled out- and in- degrees are meant to give information on the different types of edges coming out from (or going into) a node. Partial degree provides a metric of the multi evaluation of pairs (subject-predicate or predicate-object), while labeled degree refines the star-shaped nodes categorization. For instance, a high partial out-degree denotes that a pair (s,p) is related to multiple objects (multivalued) and a high labeled degree shows that the subject is related to multiple predicates (star-shaped node).

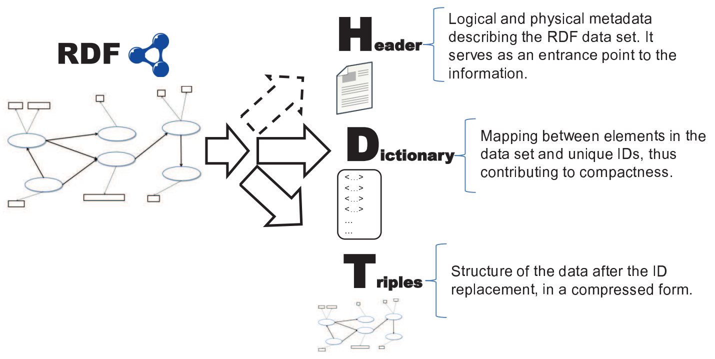
Direct out- and in-degrees complete the degree metrics for subject and objects. They indicate the cardinality of binary relations between subjects and objects disregarding the labels, *i.e.*, the columns and rows of the adjacency matrix arising when obviating the predicates.

Predicate degree constitutes an important metric for vertical partitioning technique [1], in which an index $(subject, object)$ is created for each predicate. Predicate degree reflects the number of entries for such a predicate table. In turn, predicate in-degree and out-degree refine this metric by providing a characterization of the domain and range sizes for each predicate. For instance, predicates such as *rdf:type* have a limited range (low predicate out-degree) but a great domain (high predicate in-degree).

Finally, ratios give evidence of further characteristics of RDF graphs and datasets. The subject-object ratio is a good measure of the percentage of nodes along which there are incoming and outgoing edges. These are the key edges to index, because of the different roles they play, either as subjects described elsewhere, or as objects describing other resources. Subject-predicate and predicate-object ratios show how far predicates are also used as subjects or objects. These two ratios can be used to justify the consideration of RDF as a graph or the low influence of these types of shared nodes.

Appendix Appendix A illustrates these metrics for real-world RDF datasets. The study firstly reveals that there are important differences between datasets from

[6]DBpedia: http://dbpedia.org



RDF
Header
Logical and physical metadata describing the RDF data set. It serves as an entrance point to the information.
Dictionary
Mapping between elements in the data set and unique IDs, thus contributing to compactness.
Triples
Structure of the data after the ID replacement, in a compressed form.

Figure 3: Description of HDT Components: Header-Dictionary-Triples.

different domains (*e.g* subject nodes in Uniprot appear in average in 4.27 triples whereas in Dbpedia-en it does in 12.62).

The empirical data also show the large presence of star-shaped nodes but a low frequency of multivalued pairs (s, p); the number of predicates related to a given object is very close to 1, whereas a mean of 4-5 different predicates are related with the same subject.

The ratios reveal a level of cohesion in the data; subject-object is the most frequent path constructor (subject nodes are also objects up to 61%), whereas subject-predicate and predicate-object ratios are almost negligible. Finally, power law distributions exist in both in- and out- degrees, but the skewed distributions of predicate degrees do not fit well to a power law.

All these results give insights into the RDF real-world structure and point to possible compact design models.

3. Splitting RDF in Logical Components

The skewed structure of real-world RDF data, particularly the presence of power-law distributions (see Appendix Appendix A for details), gives a starting point for designing a compact RDF structure.

In this section, we present such a compact RDF structure, called HDT, to succinctly represent the information of an RDF dataset by organizing and representing the RDF graph in terms of three components: Header, Dictionary and Triples (see Figure 3). We will show that this organization allows to represent and manage RDF data in an efficient manner. In the following, we will discuss each of these components on an abstract level as well as general uses and operations to be performed on the separate components. Practical details (*e.g.* encodings, vocabularies, etc.) are discussed in Section 4.

3.1. Header

The Header component is responsible for providing metadata about an RDF dataset. Although there are dedicated RDF vocabularies to describe metadata about datasets (*e.g.* VoiD [4]; the various annotation properties listed in the OWL vocabulary [42, Section 10]), metadata provided in the same RDF graph as the actual data causes problems, particularly making difficult to automatically distinguish between data and metadata. Whereas current other serialization formats do not provide any means or even best practices on how to publish metadata along with datasets, in HDT, we make metadata a first-class citizen with a dedicated place as part of the header information.

We consider the Header as a flexible component in which the data provider may include a desired set of features. We distinguish four basic types of metadata:

- *Publication information*. Collects the metadata about the publication act such as the site of publication, dates of creation and modification, version of the dataset (which could be useful for updating notifications), language, encoding, namespaces, etc. It also includes all kind of authority information about the source (or sources) of data.

- *Dataset statistics*. When managing huge collections, one could consider including some precomputed statistics about what follows in the datasets. For instance, it could be useful to include an estimation of the parameters presented in Section 2.1, or a subset of them used in the concrete design.

- *Format information*. Collects the information about the concrete format of the RDF dataset, *i.e.*, it specifies the concrete Dictionary and Triples implementations as well as their physical locations.

- *Other information*. A provider can take into account other metadata for the understanding and management of the data.

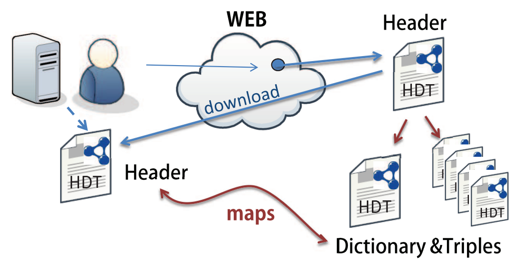


Figure 4: Example of use of the HDT Header. A consumer can download or query the Header which maps potentially distributed Dictionary and Triples components.

3.1.1. *Header Uses and Operations*

The Header serves as an entry point to the RDF dataset. Figure 4 shows a typical use case. A consumer, whether a user or machine, accesses the web page where the provider publishes the dataset. The Header is downloaded or queried online by the consumer, who is able to access 1) publication, statistical or other metadata and 2) specific format information. For instance, the consumer can see a detailed summary of the published dataset, which might be (i) distributed in several chunks, (ii) available in different formats or (iii) available under different versions. This allows the user to get: (i) the relevant chunk from the large collection and thus minimizing the exchange; (ii) the best-fitting format considering the trade-off compression ratio versus functionality; and (iii) in the adequate version.

The desired operations over the Header are as simple as the operations over a general metadata file. Publishers typically write the Header once, but it could be updated with newer information. In turn, consumers will download and access the Header locally, or they might consume it using SPARQL queries [48]. The only constraint over the metadata is that it should be machine-readable, and it should be possible to query a given type of metadata.

3.2. *Dictionary*

In general terms, a data dictionary is a centralized repository of information about data such as meaning, relationships to other data, origin, usage, and format [36]. Current RDF formats use elementary versions of dictionaries for namespaces and prefixes. This allows for the abbreviation of long and repeated strings (URIs, Literals, etc.). A good example is `‘‘http://www.w3.org/1999/02/22-rdf-syntax-ns#type’’` repeated hundreds to thousands of times in the Billion Triple dataset. Note that XML has this functionality in the form of namespaces in conjunction with *XML Base*, and several RDF formats allow abbreviations of this kind (`@base`, `@prefix` in N3 and Turtle).

Large RDF datasets are supposed to be managed by automatic processes, hence an effective replacement can be done: the Dictionary component assigns a unique ID to each element in the dataset. This way, the dictionary contributes to the goal of compactness by replacing the long repeated strings in triples by short IDs. In fact, the assignment of IDs, referred to as *mapping* [19], is usually the first step in RDF indexing. To the best of our knowledge, although there are some approaches [54, 40] which exploit the dictionary construction besides the RDF stores, the dictionary has not been proposed in any RDF representation syntax.

The Dictionary component in HDT allows multiple configurations and implementations. The sets of subjects, predicates and objects in RDF are not disjoint, thus RDF engines usually map shared elements with the same ID [7]. In turn, the order of the elements within each set could be random or sorted by some property, *e.g.* the frequency of use or the alphabetical order.

3.2.1. *Dictionary Uses and Operations*

The main goal of the Dictionary is to contribute to compactness by the assignation of a unique ID to each element in the dataset. Thus, the use of the dictionary implies two important and minimum operations:

- *locate(element)*: returns a unique identifier for the given *element*, if it appears in the dictionary.
- *extract(id)*: returns the element with identifier *id* in the dictionary, if it exists.

In addition, the dictionary might help in query evaluation and resolution. For instance, FILTER operations in SPARQL restrict the final result by a given condition. This condition usually refers to a regular expression, language or datatype selection which can be evaluated firstly over the Dictionary. Note that the elements satisfying the condition will delimit a range to search in the structure of triples.

3.3. *Triples*

By means of the Dictionary component, an original RDF triple can now be expressed as three IDs, replacing each element in a triple with the reference to the dictionary. The Triples component compacts the information by transforming a stream of strings into a stream of IDs.

In addition to its compacting feature, the Triples component is the key component to access and query the RDF graph information. The Triples component allows diverse configurations and implementations, which might exploit the trade-off between the compression ratio and the natively supported operations over the triples.

The format for RDF Triples should be designed to optimize the common operations and uses of them. We distinguish here the fundamental ones:

L0 *Exchange.* At fundamental level, an RDF Triples component serves to compact the set of RDF statements, optimizing the objective of exchange. It might include functionalities to exchange only a part of the entire graph.

L1 *Basic Search.* An important foundation for any search over RDF triples are *triple patterns*, which are templates of RDF triples where one or more element of the triple can be a variable. RDF Triples components should be able to resolve efficiently as many types of triple patterns as possible.

L2 *Join Resolution.* Joins are one of the most expensive operations in RDF queries. They imply matching two or more triples patterns which share one or more variables. RDF Triples components should support the most common types of joins (*e.g.* Subject-Subject, Object-Object, Subject-Object, etc.).

L3 *Complex querying.* Ideally, the engine should be able to answer efficiently any SPARQL query. This involves addressing many other operators and modifiers, such as UNION, OPTIONAL, as well as query evaluation optimization techniques.

The efficient indexing of the triples structure is one of the keys for good RDF query performance. However, the RDF information is exchanged in verbose, plain formats and these indexes need to be created locally by an RDF store. The `HDT` Triples component is designed to encourage the exchange of compressed triples structures which can be queried without the need of decompression (cf. [5]).

4. Practical Deployment of `HDT`

The `HDT` representation is flexible, allowing diverse implementations of the Header, Dictionary and Triples components. This feature permits to optimize different parameters for specific applications, *e.g.* compression size, compression/decompression times or querying operations over the triples.

In this section, we present practical `HDT` component implementations for efficient RDF publication and exchange. The optimization in this case is focused on compressibility and triple pattern resolution.

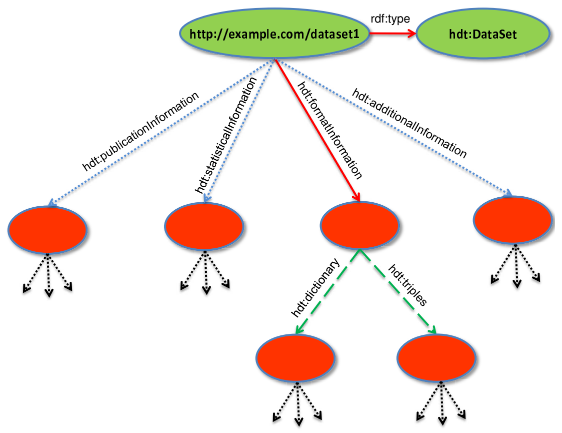

http://example.com/dataset1
rdf:type
hdt:DataSet
hdt:publicationInformation
hdt:statisticalInformation
hdt:formatInformation
hdt:additionalInformation
hdt:dictionary
hdt:triples

Figure 5: The structure of the proposed `HDT` Header is shown. It is itself an RDF graph in tree form whose nodes describe the publication, statistics, format and other type of information of the dataset.

4.1. *Extending VoiD for Header*

The Header component is expressed in plain RDF, and the use of standard vocabularies, such as VoiD, is strongly recommended. VoiD is an RDF Schema vocabulary for expressing metadata about RDF datasets, a mechanism for publishers to report their data, and an entrance point for consumers to discover and retrieve datasets. This section gives the details of our extension of the VoiD vocabulary for the particularities of binary RDF in `HDT` format, improving the publishing and exchanging RDF data at large scale. We refer to `hdt` as the vocabulary extension, with the namespace `http://purl.org/HDT/hdt#hdt`.

4.1.1. *The Header Structure*

The proposed extension assumes that the Header is an RDF graph. The triples of this graph should contain metadata about a publication together with the information required to retrieve and process the represented RDF graph in a machine processable format. It may also contain other metadata not related to these processes.

The Header of the `HDT` is described in RDF as an `hdt:Dataset` element, which is a subclass of `void:Dataset` (`rdfs:subClassOf` property). Thus, the Header can make use of VoiD properties to describe the `HDT` dataset in a standard way.

The structure of the proposed Header is represented in Figure 5. A Header must include at least one resource of type `hdt:Dataset`, described by four top-level statements (containers), which are described in the following subsections. Exactly one format metadata description must be present in order to retrieve the dataset, and `hdt:dictionary` and `hdt:triples` definition (or subproperties of them) are required.

```
@prefix void: <http://rdfs.org/ns/void#>.
@prefix dc: <http://purl.org/dc/terms/>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix hdt: <http://purl.org/HDT/hdt#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix swp: <http://www.w3.org/2004/03/trix/swp-2/>.

<http://example.org/ex/DBpediaEN> a hdt:Dataset;
                                    hdt:publicationInformation  _:publication;
                                    hdt:statisticalInformation  _:statistics;
                                    hdt:formatInformation       _:format;
                                    hdt:additionalInformation   _:additional.

_:publication     dc:issued "2010-10-01";
                  dc:license <http://www.gnu.org/copyleft/fdl.html>;
                  dc:publisher [  a foaf:Organization;
                                  foaf:homepage <http://example.org/theCompany>];
                  dc:source <http://downloads.dbpedia.org/3.5.1/en/>;
                  dc:title "DBpediaEN";
                  void:sparqlEndpoint <http://example.org/ex/DBpedia/sparql>.

_:statistics      void:triples "7";
                  void:properties "4" .

_:format          hdt:dictionary _:dictionary;
                  hdt:triplesBitmap _:triples.

_:dictionary      dc:format "application/x-gzip";
                  hdt:dictionaryEncoding "utf8";
                  hdt:dictionaryNamespaces [hdt:namespace [hdt:prefixLabel "dbpedia";
                                                           hdt:prefixURI "http://dbpedia.org/resource/"]];
                  hdt:dictionaryOrder <hdt:alphabetical_order>;
                  hdt:dictionarySeparator "\\2";
                  hdt:fileLocation <http://example.org/ex/DBpediaEN.dic>.

_:triples         hdt:predicateStream [dc:format "application/octet-stream";
                                       hdt:fileLocation <http://example.org/ex/DBpediaEN.t3_p>;
                                       hdt:IDCodification "32"];
                  hdt:predicateBitmap [hdt:fileLocation <http://example.org/ex/DBpediaEN.t3_pb>];
                  hdt:objectStream    [hdt:fileLocation <http://example.org/ex/DBpediaEN.t3_o>;
                                       hdt:IDCodification "32"];
                  hdt:objectBitmap    [hdt:fileLocation <http://example.org/ex/DBpediaEN.t3_ob>].

_:additional      swp:signature "AZ8QWE..."^^<xsd:base64Binary>;
                  swp:signatureMethod <swp:JjcC14N-md5-xor-rsa>.
```

Figure 6: A Header in `HDT`.

Figure 6 shows a Header example in Turtle syntax [9] for an RDF graph such as the one from Figure 2. These are the elements in the Header:

1. Publication Metadata. (`hdt:publicationInformation`) It groups the statements about the publication act, *i.e.*, the process of making RDF data publicly available for several purposes and users.

In addition to VoiD properties, the use of well-known vocabularies (*e.g.* Dublin Core[7] for basic metadata or WAIVER for rights[8]) is highly recommended.

2. Statistical Metadata. (`hdt:statisticalInformation`) Publishers include statistical statements about the data which can provide a fast overview of the dataset complexity as well as serving for the final application (*e.g.* in visualization and summary). VoiD property set includes statistics such as the number of RDF triples of the dataset, or the number of described entities. RDF-Stats [37] histograms, semantic statistics with SDMX [22] or the RDF Data Cube Vocabulary[9], and the previous metrics might also be included.

3. Format Metadata. (`hdt:formatInformation`) It groups the statements specifying the concrete Dictionary and Triples component representation as well as their physical location. This metadata must be present in order to retrieve the dataset, and it is required to contain an `hdt:dictionary` and `hdt:triples` definition (or subproperties of them). This metadata de-

[7]http://www.dublincore.org/
[8]http://vocab.org/waiver/terms/

[9]http://www.w3.org/TR/2012/WD-vocab-data-cube-20120405/

pends on the concrete implementation of both Dictionary and Triples. Figure 6 shows the configuration for Plain Dictionary and Bitmap Triples.

4. Additional Metadata. (`hdt:additionalInformation`)
It contains all kind of additional information given by the publisher, *e.g.* annotations, or a signature as shown in Figure 6.

4.2. Dictionary Encodings. Plain Dictionary

A practical encoding for the dictionary component is proposed as follows, referred to as Plain Dictionary (`hdt:dictionaryPlain`). Four subsets of elements are considered, mapped as follows (in an RDF graph G with S_G, P_G, O_G different subjects, predicates and objects):

1. *Common subject-objects*, denoted as the set SO_G, are mapped to $[1, \ |SO_G|]$.
2. The *non common subjects*, $S_G - SO_G$, are mapped to $[|SO_G| + 1, |S_G|]$.
3. The *non common objects*, $O_G - SO_G$, are mapped to $[|SO_G| + 1, |O_G|]$.
4. *Predicates* are mapped to $[1, \ |P_G|]$.

Figure 7 shows an example of these four sets for the RDF graph of Figure 2. Note that a given ID can belong to different sets, but the disambiguation of the correct set is trivial when we know that the ID in a triple is placed as a subject, a predicate or an object. A similar partitioning is taken in some RDF indexing approaches [7].

The subject-object ratio defined in Section 2.1, α_{s-o}, characterizes the proportion of the subject-object set in the dictionary, composed of nodes with out-degree and in-degree greater than 0, $deg^-(a), deg^+(a) > 0$. In those datasets with a noticeable value of α_{s-o}, common subject-object identification reduces the dictionary size versus a disjoint assignment. The set of predicates are treated independently because of their low number and the infrequent overlapping with other sets. Due to the sequential mapping of each set, the dictionary only has to include the strings, assuming an implicit order of IDs and some form of distinction between sets.

The physical Dictionary consists of a list of strings matching the mapping of the four subsets, in order from (1) to (4), as shown in Figure 8. A reserved character (we use '\2 'by default) is appended to the end of each string and each section to delimit their size.

4.3. Triples Encodings

We provide three implementations for Triples component encoding (plain, compact, bitmap) as shown in Figure 9 for the given graph in Figure 2.

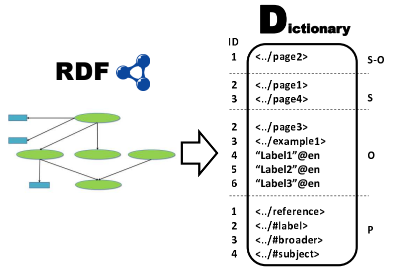


Figure 7: HDT Dictionary example.

```
<http://dbpedia.org/resource/page2> \2\2 <http://dbpedia.org/resource/page1
>\2 <http://dbpedia.org/resource/page4> \2\2 <http://dbpedia.org/resource/
page3>\2 <http://example.org/example1> \2 "Label1"@en \2 "Label2"@en \2\2
<http://dbpedia.org/property/reference> \2 <http://www.w3.org/2000/01/rdf-
schema#label> \2 <http://www.w3.org/2004/02/skos/core#broader> \2 <http://
www.w3.org/2004/02/skos/core#subject> \2\2
```

Figure 8: HDT physical Dictionary component example.

4.3.1. Plain Triples (`hdt:triplesPlain`)

This is the most naive approach in which only the ID substitution is performed, as shown in Figure 9. The physical file contains three IDs per triple.

4.3.2. Compact Triples (`hdt:triplesCompact`)

This option implies a triple sorting by subject and the creation of predicate and object adjacency lists.

Adjacency List is a compact data structure that facilitates managing and searching. For example, the set of triples: $\{(s, p_1, o_{11}), \cdots, (s, p_1, o_{1n_1}), (s, p_2, o_{21}), \cdots (s, p_2, o_{2n_2}),$

$\cdots (s, p_k, o_{kn_k})\}$

can be written as the adjacency list:

$$s \to [(p_1, (o_{11}, \cdots, o_{1n_1}), (p_2, (o_{21}, \cdots, o_{2n_2})),$$
$$\cdots (p_k, (o_{kn_k}))].$$

Turtle (and hence N3) allows such generalized adjacency lists for triples. For example the set of triples $\{(s, p, o_j)\}_{j=1}^{n}$ can be abbreviated as $(s \ p \ o_1, \cdots, o_n)$.

The Triples component contains a compact adjacency list representation. First, a subject ordered grouping is performed, that is, triples are reorganized in an adjacency list, in sequential order of subject IDs. Due to this order, an immediate saving can be achieved by omitting the subject representation, as we know the first list corresponds to the first subject, the second list to the following, and so on.

Then, the representation is split into two coordinated streams of Predicates and Objects. The first stream of Predicates corresponds to the lists of predicates associated with subjects, maintaining the implicit grouping order. The end of a list of predicates implies a change of subject, and must be marked with a separator, the non-assigned zero ID. The second stream (Objects) groups the lists of objects for each pair (s, p). These pairs are

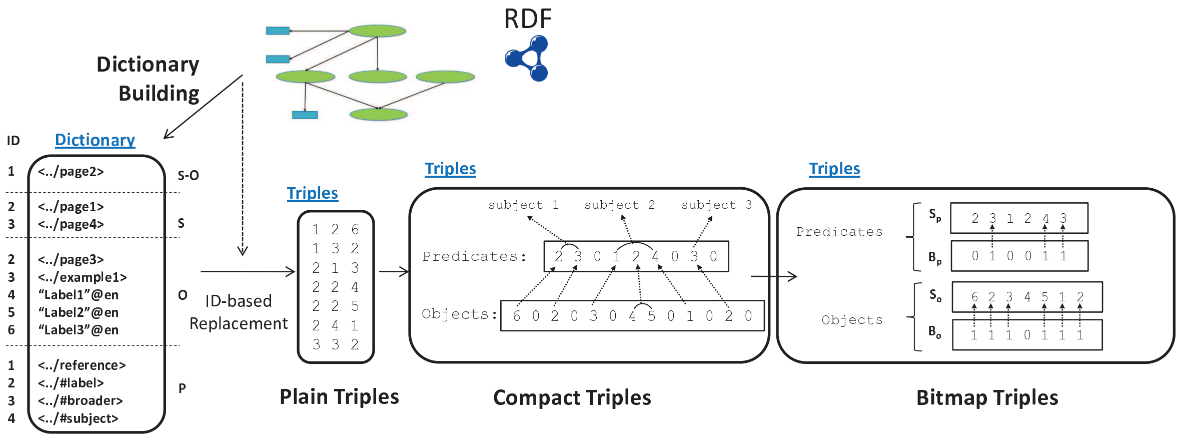


Figure 9: Three possibilities of triple representations.

formed by the subjects (implicit and sequential), and coordinated predicates following the order of the first stream. In this case, the end of a list of objects (also marked in the stream with the non-assigned zero ID) implies a change of (s, p) pair, moving forward in the first stream processing.

Thus, the compact triple representation is supported by two streams: (1) a predicate stream in which the predicate lists are separated by 0s (i-th list belongs to i-th subject) and (2) an object stream in which the object lists are separated in the same way (j-th list belongs to the j-th subject/predicate pair in the former stream).

The parameters in Section 2.1 characterize the streams. Labeled out-degree, $degL^{-}(s)$, indicates the length of the list of predicates for a subject s. Therefore, the maximum and mean lengths of the lists in *Predicates* are given by $degL^{-}(G)$ and $\overline{degL^{-}}(G)$ respectively. Symmetrically, partial out-degree, $deg^{--}(s,p)$ gives the size of the corresponding list in *Objects*. Maximum and mean values, $deg^{--}(G)$ and $\overline{deg^{--}}(G)$ characterize the *Objects* stream.

This leads to a compact ID-based triple representation in which the classical three-dimensional view of RDF has been reduced into two by the coordinated streams, considering implicit the third dimension of subjects.

4.3.3. *Bitmap Triples (*`hdt:triplesBitmap`*)*

In Compact Triples, two coordinated ID-based streams, *Predicates* and *Objects*, *draw* the RDF graph, representing the triples with an implicit subject-grouping strategy. Both streams can be seen as sequences of non-negative integers in which `0`-values mark the endings of predicate and object adjacency lists respectively. This means that positive integers represent predicates and objects, whereas `0`'s are auxiliary values embedded in each stream to represent, implicitly, the graph structure. Bitmap Triples implementation splits both parts in order to improve the `HDT` usability.

This encoding extracts the 0's out of the predicate and object streams of the Compact Triples representation. The graph structure is indexed with two bitsequences ($\mathcal{B}_p$ and $\mathcal{B}_o$, for predicates and objects) in which 1-bits are used to mark the end of an adjacency list. This transformation is shown in Figure 9. On the one hand, Predicates=$\{2,3,0,1,2,4,0,3,0\}$ evolves to the sequence $\mathcal{S}_p = \{2,3,1,2,4,3\}$ and the bitsequence $\mathcal{B}_p = \{010011\}$ whereas, on the other hand, Objects= $\{6,0,2,0,3,0,4,5,0,1,0,2.0\}$ is reorganized in $\mathcal{S}_o = \{6,2,3,4,5,1,2\}$ and $\mathcal{B}_o = \{1110111\}$.

The triples structure can be interpreted as follows. The i-th `1`-bit in $\mathcal{B}_p$ marks the end of the predicate adjacency list for the i-th subject (it is referred to as P_i), whereas the number of predicates in the corresponding list can be obtained by subtracting the positions of two consecutive `1`-bit (we always consider that positions are numbered from "1"). For instance, the second `1`-bit in $\mathcal{B}_p$ marks the end of the predicate adjacency list for the second subject (P_2). There are three positions between the second and the first `1`-bit in $\mathcal{B}_p$. Thus, P_2 contains three predicates, which are represented by the third, fourth and fifth IDs in $\mathcal{S}_p$, hence $P_2 = \{1,2,4\}$.

Data in $\mathcal{S}_o$ and $\mathcal{B}_o$ are related in the same way. The j-th `1`-bit in $\mathcal{B}_o$ marks the end of the object adjacency list for the j-th subject/predicate pair. This predicate is represented by the j-th position in $\mathcal{B}_p$ and it is retrieved from the j-th position of $\mathcal{S}_p$. For example, the third `1`-bit in $\mathcal{B}_o$ refers the end of the object adjacency list for the third predicate in $\mathcal{S}_p$ which is related to the second subject as we have previously explained. Thus, this adjacency list stores all objects o in triples $(2,1,o) \in G$.

Operations. Each element in $\mathcal{S}_p$ and $\mathcal{S}_o$ is encoded, respectively, with a fixed-length code of $\log(|P_G|)$ and

Algorithm 1 `Check&Find` operation for a triple (s, p, o)

```
1: $begin \leftarrow \mathbf{select_1}(B_p, s-1)+1$;
2: $end \leftarrow \mathbf{select_1}(B_p, s)$;
3: $size_{P_s} \leftarrow end - begin + 1$;
4: $P_s \leftarrow \mathbf{retrieve}(S_p, begin, size_{P_s})$;
5:
6: $plist \leftarrow \mathbf{binary\_search}(P_s, p)$;
7: $pseq \leftarrow begin + plist - 1$;
8:
9: $begin \leftarrow \mathbf{select_1}(B_o, pseq-1)+1$;
10: $end \leftarrow \mathbf{select_1}(B_o, pseq)$;
11: $size_{O_{sp}} \leftarrow end - begin + 1$;
12: $O_{sp} \leftarrow \mathbf{retrieve}(S_o, begin, size_{O_{sp}})$;
13:
14: $plist \leftarrow \mathbf{binary\_search}(O_{sp}, o)$;
```

$\log(|O_G|)$ bits, by considering that the dataset comprises $|P_G|$ and $|O_G|$ different predicates and objects. The bitsequences used to represent $\mathcal{B}_p$ and $\mathcal{B}_o$ make use of *succinct structures*. They are able to support `rank/select` operations over a sequence $\mathcal{S}$ of length n drawn from an alphabet $\Sigma = \{0, 1\}$:

- `rank`$_a$`(S,i)` counts the occurrences of a symbol $a \in \{0, 1\}$ in $\mathcal{S}[1, i]$.
- `select`$_a$`(S,i)` finds the i-th occurrence of symbol $a \in \{0, 1\}$ in $\mathcal{S}$. In practice, $\mathbf{select_a}(S, 0) = 0$;

This problem has been solved using $n + o(n)$ bits of space while answering the queries in constant time [20]. We choose the *González, et al.* [28] approach to implement $\mathcal{B}_p$ and $\mathcal{B}_o$. This adds 5% of extra space to the original bitsequence lengths, and achieves constant time for the `select/rank` operations, which constitutes the basis for accessing to the structure of the graph.

Bitmaps Triples representation allows a retrieval strategy able to take advantage of the structure indexed in $\mathcal{B}_p$ and $\mathcal{B}_o$, accessible by fast `rank/select` operations. Algorithm 1 shows a `Check&Find` operation for a triple (s, p, o) over Bitmaps Triples.

Lines `1-4` describe the steps performed to retrieve the predicate adjacency list for the subject s (P_s). First, we obtain its size by locating its begin/end positions in $\mathcal{B}_p$. Next, we `retrieve` its sequence of $size_{P_s}$ predicate IDs from $\mathcal{S}_p$. Once P_s is available, we need to identify the position ($pseq$) where s and p are related in $\mathcal{S}_p$. Lines `6-7` describe it. First, p is located in P_s with a `binary_search`, and, next, this local position, $plist$, is used to obtain its global position in $\mathcal{S}_p$, $pseq$, by adding $begin$ to $plist$. In this step, the object adjacency list for s, p (O_{sp}) can be retrieved because it is indexed through the pseq$-th$ predicate. O_{sp} is retrieved (lines `9-12`) similarly to P_s, considering $\mathcal{B}_o$ and $\mathcal{S}_o$. Finally, o is located with a `binary_search` on O_{sp}.

The cost of the `Check&Find` operation for a triple (s, p, o) is $O(size_{P_s} + size_{O_{sp}})$, assuming at most $size_{P_s} = degL^-(s)$ and $size_{O_{sp}} = deg^{--}(s, p)$. The distribution of lists assures an amortized cost in $(\overline{degL^-(G)} + \overline{deg^{--}(G)})$. Note that this operation does not just find the required triple (s, p, o), but also the triples $(s, p, z) \in G$. Besides, P_s contains all predicates from s, so the next operations on triples from s begin the `Check&Find` operation by identifying the position of p in $\mathcal{S}_p$ (from line `6`).

Efficient access is obtained through `Check&Find`. If a triple $(s, p, o) \notin G$, it can be detected in step `6` (the predicate p is not in the predicate adjacency list for s: `S`$_p$) or in step `14` (the object o is not in the object adjacency list for s and p: `O`$_{sp}$). On the contrary, if $(s, p, o) \in G$, once the triple is found, the strings associated with s, p, and o can be retrieved from the dictionary. Note that the aforementioned Plain Dictionary is only intended for compact purposes, hence it should be loaded into a functional structure, such as a Hash table, B-tree or any other structure optimized for dictionary management [40].

Furthermore, the SPARQL query language for RDF can make use of some interesting features of Bitmap Triples, as follows:

- Algorithm 1 can answer basic ASK queries of SPARQL for patterns (s,p,o), (s,?p,?o) and (s,p,?o).
- Algorithm 1 can response basic CONSTRUCT query of SPARQL for simple WHERE patterns (s,p,o), (s,?p,?o) and (s,p,?o).The result is a RDF `HDT` graph.

The S-P-O Adjacency List order must be assumed. The response patterns vary for alternative representations S-O-P, P-S-O, P-O-S, O-P-S and O-S-P Adjacency Lists.

5. Evaluation of HDT

This section evaluates the size and performance of the `HDT` deployment presented in the previous section. First, we measure the size of the `HDT` Dictionary and Triples to show its good compact ratio performance. Then, we evaluate the scalability of `HDT` based on the implementation of Plain Dictionary and Bitmap Triples. Finally, we evaluate triple pattern queries performance.

These tests were performed on a Debian 4.1.1 operating system, running on a computer with an AMD Opteron(tm) Processor 246 at 2 GHz and 4 GB of RAM.

dataset	`Plain Dictionary`	Triples		
		`Plain`	`Compact`	`Bitmap`
Geonames	*12.54%*	9.33%	4.71%	2.91%
Wikipedia³	*4.53%*	7.82%	2.45%	2.09%
Dbtune	*10.34%*	6.93%	3.95%	2.53%
Uniprot	*11.08%*	12.05%	6.16%	4.05%
Dbpedia-en	*14.10%*	7.12%	3.53%	2.66%

Table 1: Compact results.

We used a g++ 4.1.2 compiler with `-09` optimization. This experimentation was run on the datasets described in Appendix Appendix A. For the evaluation, we consider a Header in Turtle syntax such as the one from Figure 2. Note that the size of the Header (a few KB at most) is negligible at large scale.

5.1. Dictionary and Triples compact ability

Table 1 shows the compact ratios of each proposed component in `HDT` over the original N-Triples format (one triple per line). It is very interesting to note that Plain Dictionary and Plain Triples have a comparable ratio, hence the need of improving both components to improve the final result. Compact Triples clearly outperforms Plain Triples, achieving ratios around 4% and up to 2.45% of the original size. Bitmap Triples is the most compact solution for the triples, obtaining ratios around 3% and up to 2.09% over the original size.

In addition to its effectiveness, Bitmap Triples also overcomes Compact Triples in its ability for direct accessing to the compressed data, so we will use Bitmap Triples implementation in all remaining experiments.

Regarding the dictionary, please note that it takes five times more space than the Bitmap Triples. Moreover, the current dictionary does not provide SPARQL boosting operations. These insights encourage the use of compact RDF dictionary implementations [40].

5.2. HDT-Compress

`HDT` (referred to as `Plain-HDT` henceforth) achieves a considerable size reduction of the RDF dataset by means of the Plain Dictionary and the Bitmap Triples configurations. This provides a clean publication and efficient exchange ratios. However, `Plain-HDT` is even more compressible with very little effort. We test HDT compressibility with a particular deployment called `HDT-Compress`. This deployment makes specific decisions:

- Header: We keep the Header component in plain form as it should always be available to any receiving agent for processing and its size is negligible.

- Dictionary: We take advantage of repeated prefixes in URIs, specific n-gram distributions in literals, etc. We choose a predictive high-order compressor, PPM [21], which identifies this type of redundancy to improve the encoding of the dictionary.

- Triples: The set of bitmap triples compression is independently attempted on each structure. On the one hand, $\mathcal{S}_p$ comprises an integer sequence drawn from $[1, |P_G|]$. A `Huffman` [35] code is used to compress it. On the other hand, the compression of $\mathcal{S}_o$ (drawn from $[1, |O_G|]$) takes advantage of the power-law distribution of objects (see the right dispersion graph in Figure A.17) through a second `Huffman` code. Finally, we hold a plain representation for bitsequences because of the small improvement obtained with specific techniques for bitsequence compression.

We chose `shuff`[10] and `ppmdi`[11] to implement, respectively, the `Huffman` and `PPM`-based encoding.

Table 2 compares `HDT` against three well-known universal compressors. We chose `gzip` as a dictionary-based technique on LZ77, `bzip2` based on the Burrows-Wheeler Transform, and `ppmdi` as a predictive high-order compressor.

The most effective universal compressors for all datasets are `ppmdi` and `bzip2` which achieve ratios of around 4% and 5% respectively. A very interesting result shows that `Plain HDT` is able to outperform `gzip` for the Wikipedia³ dataset. This demonstrates the previously cited ability of `HDT` to obtain compact representations of RDF.

`HDT-Compress` achieves the most effective results with ratios between $2-4\%$ for the considered datasets. This implies reductions between $3-4$ times with respect to `Plain HDT`, and consequently proportional improvements on exchanging processes. In turn, `HDT-Compress` outperforms universal compressors by improving the best `ppmdi` results between $20-45\%$.

[10]http://www.cs.mu.oz.au/∼alistair/mr_coder
[11]http://pizzachili.dcc.uchile.cl/experiments.html

dataset	Triples (millions)	Size (GB)	`HDT`		Universal Compressors		
			`Plain`	`Compress`	`gzip`	`bzip2`	`ppmdi`
Geonames	9.4	1.00	*15.45%*	**3.16%**	7.67%	5.35%	4.80%
Wikipedia³	47	6.88	*6.62%*	**2.22%**	6.97%	5.11%	4.10%
Dbtune	58.9	9.34	*12.86%*	**1.83%**	9.67%	6.59%	4.62%
Uniprot	72.5	9.11	*15.13%*	**3.54%**	13.22%	7.93%	6.83%
Dbpedia-en	232.5	33.12	*16.77%*	**3.89%**	10.36%	7.80%	6.64%

Table 2: Compression results.

5.3. *Scalability Evaluation*

We study the `HDT` scalability in two correlated aspects. First, we evaluate the `HDT` performance with incremental size of a dataset. Then, we test `HDT` components compact ability in a wide range of different datasets to show that `HDT` results can be extrapolated to general fields of application.

5.3.1. *Incremental Size*

We study the `HDT` performance with incremental size of the Uniprot dataset, from 1 to 40 million triples. This is shown in Figure 10. The left table studies the `HDT` evolution of effectiveness. As can be seen, the ratios go between $14-15\%$ for `Plain HDT`, and around 3.5% for `HDT-Compress` (the percentage is always given with regard to the original file size). This ensures `HDT` effectiveness by considering that its compression ratios do not strongly depend on the dataset size, although best numbers are achieved for larger datasets.

The right graph of Figure 10 shows relevant times for `HDT`. On the one hand, the *creation* time stands for the time required to transform an RDF dataset (from plain N-Triples) into `HDT`. This process is only performed once at publishing and shows a sublinear growth. On the other hand, after the *loading* time, the minimum information required for `HDT` management is in memory and available to be accessed with the `Check&Find` mechanism (Algorithm 1). As can be seen, this time is only a very small fraction ($\approx 3\%$) of the creation one. Additionally, symmetrical *compression* and *decompression* times are achieved with `HDT-Compress`. In absolute terms, both compression and decompression times are slightly worse than the loading ones.

5.3.2. *Multiple datasets*

We choose the dataset from the Semantic Web Challenge 2010[12], called Billion Triples, which contains ($\sim$3.2 billion statements). The Data is collected from Sindice, Swoogle and others, given in N-Quads [23] format. We select the four hundred largest datasets by grouping the triples according to the host of the fourth component of the NQuad. This way, we ensure a wide range of application fields.

[12]http://challenge.semanticweb.org/

First, we study the dictionary growth with respect to the number of triples in the dataset, shown in Figure 11. Each point corresponds to a different dataset. The number of unique dictionary entries has a sublinear growth *w.r.t.* the number of triples.

Then, we evaluate different Triples orderings over the Huffman-compressed Bitmap Triples implementation. We set *SPO* as the baseline to which the other five possible orderings are compared. This comparison is shown in Figure 12: X-axis lists all the remaining orderings and Y-axis shows the proportion of the *SPO* space that each ordering requires. As can be seen, *PSO* and *POS* report the worst numbers: roughly 1.4 times and 1.24 times the space used by *SPO* respectively. This is because predicates are on the top layer of the Bitmap Triples, and since the number of different predicates is always much smaller than the number of different subjects or objects (see Table A.3), fewer items are left implicit. The better *POS* effectiveness compared to *PSO* can be explained using the predicate degrees. As shown in Table A.16, $\overline{degP^{-}}(G)$ is always smaller than $\overline{degP^{+}}(G)$, so the number of objects per predicate is smaller than the number of subjects per predicate and this results in shorter adjacency lists (by object). Moreover, only *OPS* ordering outperforms the baseline because the number of predicates per object $\overline{degL^{+}}$ is smaller than predicates per subject $\overline{degL^{-}}$, so the adjacency lists per object are shorter than lists per subject. As can be seen, $\overline{degL^{+}}$ ranges from 1.00 to 1.39 predicates per object, whereas $\overline{degL^{-}}$ ranges from 4.00 to 6.13 predicates per subject. Nevertheless, the difference is small (*OPS* requires ≈ 0.97 times the space used by *SPO*), so we think *SPO* is still a good general choice. This decision also takes into account that *SPO* is a more intuitive ordering for triples, and that queries involving subject access are massively used [6]. As we show in Table 15, graph traversals are resolved very efficiently using `Check & Find` to search the neighbors of a given subject.

We compare the dictionary and triples compress ability. Figure 13 represents the frequency of the ratio $Dictionary/Triples\ size$ when applying `HDT-Compress`. The normal distribution centered at

Triples (millions)	Size (MB)	HDT `Plain`	HDT `Compress`
1	89.07	*15.11%*	**3.73%**
5	444.71	*14.54%*	**3.48%**
10	893.39	*14.04%*	**3.27%**
20	1790.41	*14.43%*	**3.31%**
30	2680.51	*14.39%*	**3.27%**
40	3574.59	*14.34%*	**3.26%**

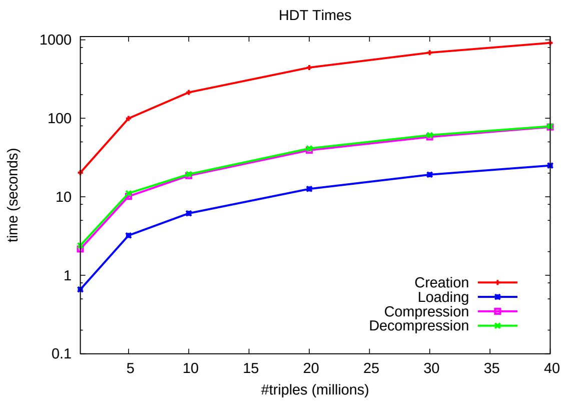


Figure 10: Performance of `HDT` (`Plain` and `Compress`) with incremental size datasets from Uniprot. The left table shows effectiveness, whereas the right figure draws significative times.

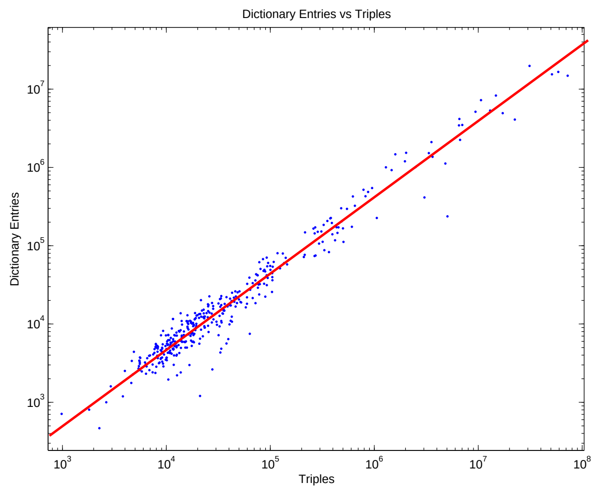


Figure 11: `HDT` Dictionary growth ($y = 0,62x^{0,97}$).

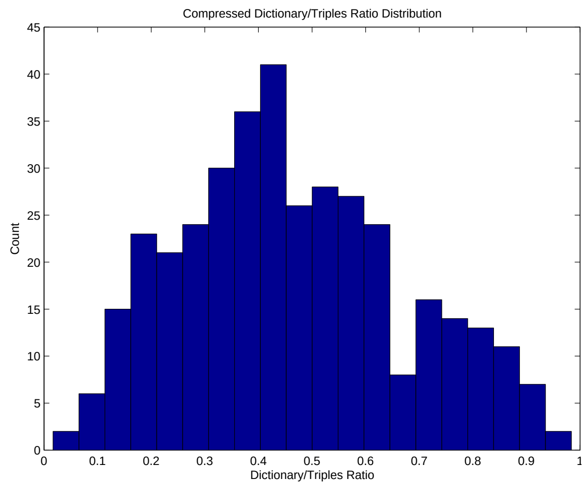


Figure 13: `HDT` Dictionary/Triples ratio distribution.

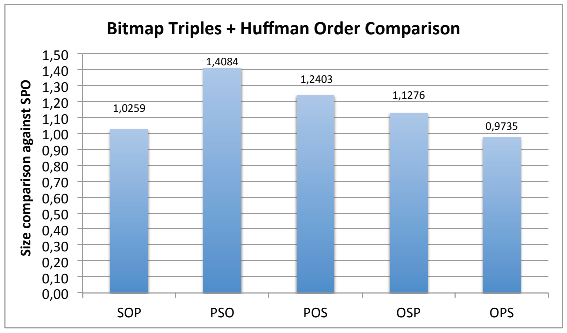


Figure 12: `HDT` Bitmap Triples (+Huffman) order comparison. Figures represent the ratio against the size of the SPO order.

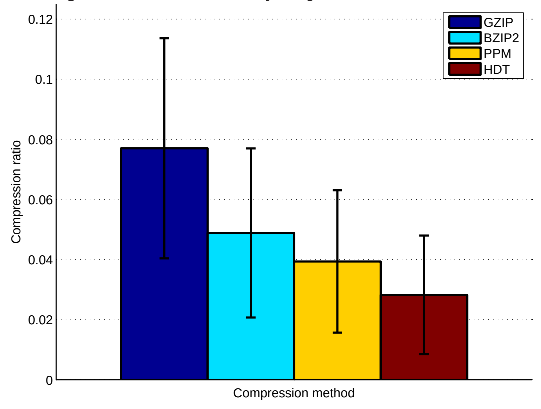


Figure 14: `HDT-Compress` results.

0.5 implies that both components similarly contribute to the final compression ratio, hence the importance of isolating and improving both parts.

Finally, Figure 14 compares the mean (over all the datasets) `HDT-Compress` ratio against the considered universal compressors, verifying the previous results.

5.4. Query Response Evaluation

Finally, we evaluate triple pattern resolution. We perform tests over the plain representation of Bitmap Triples (note that the aforementioned Huffman compressed version is only intended to show its compressibility for exchange and it is not directly queryable). We disregard also the Dictionary for a fair comparison because the optimization of the mapping operations is an orthogonal issue for most RDF engines. We experiment against state-of-the-art solutions such as RDF-3X and MonetDB which allow larger datasets to be managed. We perform on MonetDB by following the work in [51]: we create separated predicate tables and insert ID-based subject/object tuples, in this order (thus forcing a subject-object index). We also disable the dictionary in RDF-3X.

All tests are performed on the *Dbpedia* dataset and the SPARQL triple pattern queries are extracted from the log provided for the USEWOD'2011 Challenge[13]. This means that our testbed is composed by real user queries except for the patterns (S, P, O) which are directly extracted from the original dataset through a random procedure. We choose, at random, 50 different queries for each triple pattern, translated to their correspondent IDs.

Table 15 shows the performance for the considered triple patterns (averaged times, in milliseconds per pattern). Note that with our current HDT solution, the *SPO* order can only resolve efficiently the first three patterns. In order to test all patterns, the *OSP* order is used for the next two queries and *POS* for the last two ones. Thus, we do not test HDT as a full-fledged engine but we give insights of query performance over distinct orders.

First, we compare against MonetDB to evaluate the performance of HDT versus a vertical-partitioning solution. The use of HDT largely improves query times for all patterns. This is specially important for patterns with unbounded predicates because their resolution is the main weakness of the vertical-partitioning systems; for instance, (S,?P,O) takes 757 seconds/pattern in MonetDB and only 0.03 milliseconds/pattern in HDT. These large differences are an experimental evidence of the HDT possibilities for SPARQL querying.

The comparison of HDT and RDF-3X also shows interesting conclusions. HDT is the most efficient approach for almost all patterns. The only pattern where RDF-3X outperforms HDT is $(?S, P, O)$. Although $(?S, P, ?O)$ improves over RDF-3X and MonetDB, it achieves the worst time of all the triple patterns tested on HDT. It means that grouping by predicate (*POS* order) makes slower searches in the bitmap. This correlates with the high predicate degrees studied for *Dbpedia*, *i.e.*, there are many subjects and objects related with a given predicate. This fact evidences the importance of the defined RDF metrics to optimize indexes and query evaluation.

[13]http://data.semanticweb.org/usewod/2011/challenge.html

6. Related Work

HDT representation can be viewed from different points of view, uses techniques from diverse fields, and can be applied in different scenarios. It can be considered as a binary RDF format; thus, we discuss its relationship with the most relevant RDF representations. It takes advantage of the RDF structure; we review the most important findings in this direction. Regarding applications, we review and compare it with the relevant literature on RDF publication, RDF exchange and RDF indexing and querying.

6.1. RDF Representations

Today there are several syntax representations for RDF data, *e.g.* RDF/XML, N3, Turtle or RDF/JSON. None of these proposals, though, seems to have considered data volume as its primary goal.

RDF/XML [8], due to its verbosity is good for exchanging data, but only on a small scale. It includes some compacting features:

- Omitting Blank Nodes ([8], section 2.11): The attribute *rdf:parseType="Resource"* allows to implicitly create blank nodes.
- Omitting Nodes ([8], section 2.12): Under certain conditions, object nodes with string literals can be moved to property attributes, hence the subject node becomes empty.
- Abbreviating URI references ([8], section 2.14): First, a base URI attribute *xml:base* can be set. This is the base URI for resolving relative RDF URI references, otherwise the base URI is that of the current document. Then, the *rdf:ID* attribute on a node element can be used instead of *rdf:about*. This attribute must be interpreted as a relative RDF URI reference.
- Collections ([8], section 2.16): It allows an *rdf:parseType="Collection"* attribute to be defined on a property element. This provides a set of node elements related to the subject node.

Notation3 (N3 [10]) is a language which was originally intended to be a compact and readable alternative to RDF's XML syntax, optimized for reading by scripts. Thus, it reduces verbosity and represents the RDF with a simple grammar based on the plain triples philosophy. It also allows some compacting features such as abbreviations for URIs prefixes (and base URI), shorthands for common predicates and square bracket blank node syntax. One major advantage is the use of lists. For

	(S,P,O)	$(S,P,?O)$	$(S,?P,?O)$	$(S,?P,O)$	$(?S,?P,O)$	$(?S,P,O)$	$(?S,P,?O)$
RDF-3X	1.82	2.90	2.75	2.31	3.86	**2.55**	2634.86
MonetDB	26.14	50.29	677111.48	757448.11	675127.13	97.49	6397.92
HDT	**0.05**	**0.01**	**0.05**	**0.03**	**0.90**	8.31	**2108.82**

Figure 15: Query times (in ms/pattern) for simple triple patterns

instance, repetition of another objects for the same previous subject and predicate using a comma "," and repetition of another predicate for the same subject using a semicolon ";".

Turtle [9] is a more compact and readable alternative. It is intended to be compatible with, and a subset of, N3, thus it inherits its compact features, *e.g.* the abbreviation of RDF collections. N-Triples[14] is also a subset of N3, restricting to only one triple per line, using hardly any syntactic sugar. It simplifies the parsing process at the expense of avoiding compact structures.

RDF/JSON [2] resembles Turtle, with the advantage of being coded in a language easier to parse and more widely accepted in the programming world. It is intended to be easy for humans to read and write and easy for machines to parse and generate.

Although most of these formats present features to "abbreviate" constructions like URIs, groups of triples, common datatypes or RDF collections, the compactness of the representation definitely was not the main concern of their design. Finally, Sterno [55] is designed as a subset of Turtle for optimizing parallel I/O. Although it collaterally addresses some notion of initial metadata and compactness (*e.g.* all prefix declarations must occur at the beginning of a document and a Lempe-Ziv compression over Sterno is evaluated), its main purpose is to allow parallel processing (divisibility) disregarding publication facilities as well as native query support.

6.2. RDF Structure

RDF is a cornerstone in the Web of Data. It provides the generic graph-based data model used to structure and link data that describes things in the world [12]. Currently, the Web of Data comprises very large RDF datasets from diverse fields like as bioinformatics, social networks or geography, among others. RDF adoption is becoming incredible important. However, works studying its global functioning and structure are scarce.

Although power-law[15] distribution validation in RDF data remains an open field, in practice it is assumed as a common characteristic of RDF real-world data. Ding and Finn [25] reveal that Semantic Web graphs fit power-law distribution within some metrics such as the size of documents and term frequency use; most terms are described through few triples. Regarding the use of an RDF schema (RDFS[15]), the space of instances is sparsely populated, since most classes and properties have never been instantiated. By crawling the Web, Oren [46] comes to similar conclusions, showing that resources (URIs) in different documents fit to a power-law distribution. Theoharis [53] studies these properties for Semantic Web schemas, RDFS and OWL [41]. Similar distribution is found in the descendants of a class, as well as other schema features, such as the existence of few classes interconnecting schemas, or non-balanced hierarchies. The presence of star and chaining nodes has been also described in data and queries (star- and chain-shaped join queries) [43, 44]. This schema analysis has contributed to synthetic schema generation for benchmarking [53].

These results motivate the application to RDF of the well-known Web distribution, where power-law is present in successors list of a given domain, playing an important role in Web graphs compression [14, 18].

RDF compression capabilities have been studied [27] but have not been applied in a concrete format or implementation. The situation is not better for splitting RDF into components. Neither RDF/XML nor N3 (and their subsets Turtle and N-Triples) have the basic constructors to design modular files. To the best of our knowledge, none of these techniques have been applied in the design of RDF datasets.

There is little work on the design of large RDF datasets. There have been projects discussing design issues of RDF[16], and a working group on design issues of translation from relational databases to RDF[17]. However, none of these works have touched the problem of RDF publication and exchange at large. The project that is currently systematically addressing the issue of publication of RDF at large, Linked Data, is beginning to face some of these issues.

[14]http://www.w3.org/TR/rdf-testcases/#ntriples

[15]A power law is a function with scale invariance, which can be drawn as a line in the log-log scale with a slope equal to a scaling exponent, *e.g.* $f(x) = ax^{-\beta}$, thus $f(cx) \propto f(x)$, with a, c, β constants.

[16]Best Practices Publishing Vocab. W3C WG: www.w3.org/2001/sw/BestPractices/, and the Wordnet case www.w3.org/2001/sw/BestPractices/WNET/wn-conversion.html

[17]http://www.w3.org/2001/sw/rdb2rdf/

6.3. *Applications on RDF*

At this point in the evolution of the Web, RDF can actually be used in a wide variety of fields and applications. We mainly focus on (i) publication, (ii) exchange and (iii) indexing and querying.

6.3.1. *Publication*

An increasing number of data is currently published adopting RDF and the Linked Data principles [11]. Linked Open Data (LOD) cloud is estimated in more than 31 billion RDF triples and half a billion links. Furthermore, Open Data and Open Government movements have also been gaining momentum by publishing government data in standards format such as RDF. For instance, the United States government through the data.gov site hosts billions of RDF triples in several RDF datasets.

The Vocabulary of Interlinked datasets (VoiD [3]) aims to bridge data publishers and data users, so that publishers can distribute the datasets (as a RDF dump, SPARQL Endpoints, etc.) and users can discover and use identified datasets given certain attributes. It provides a vocabulary and a set of instructions that allow the discovery and usage of linked datasets.

Semantic Sitemaps [24] support efficient semantic datasets discovery and high-performance retrieval. It is based on extending the traditional Sitemap Protocol with new XML tags for describing the presence of RDF data (and to deal with specific RDF publishing needs).

6.3.2. *Exchange*

To the best of our knowledge, few works focus on RDF exchange. Actually, RDF datasets tend to be published and exchanged within plain RDF formats, such as RDF/XML, N3, Turtle and JSON. General compressors (typically gzip) are also used over these plain formats in order to reduce the final size.

A recent work [27] shows that big RDF datasets are highly compressible due to the structure of RDF graphs (power law), organization of URIs and RDF syntax verbosity. This work also approaches basic RDF data compression using (i) direct compression, (ii) adjacency list compression and (iii) an RDF division into the dictionary of elements and the triples, substituting for each element the corresponding number assignation in the dictionary. The last technique is also carried out in [54] using MapReduce and distributed algorithms to boost the efficiency of large RDF data compression and decompression. It achieves linear scalability regarding the input size and number of nodes.

RDF/XML is a valid XML format and thus XML interchange formats might be used. For instance, the Efficient XML Interchange Format (EXI [50]) is a compact representation for XML. It is based on efficient encodings of XML event streams using a grammar-driven approach. The stream of events are represented using variable length codes.

6.3.3. *Indexing and Querying*

RDF is a logical data model not limited by its physical storage or indexing technologies. However, these procedures are strongly related with the later querying process, which is typically performed by SPARQL queries. SPARQL [48] is a declarative language for extracting information from RDF graphs. It provides graph pattern matching facilities allowing to bind variables to elements in the RDF graph. In addition, SPARQL provides a series of operators (namely SELECT, AND, FILTER, OPTIONAL, and UNION) offering high expressiveness for structured queries.

Several RDF indexes and RDF storages explore efficient SPARQL resolution methods. Some approaches store RDF in a relational database and perform SPARQL queries through SQL, such as Jena-TDB [57], Virtuoso [26], and the column-oriented databases C-Store [52] and Monet-DB [51]. Two basic policies are considered to transform RDF into a relational representation (see [49] for an experimental performance comparison): (1) storing all triples in a large 3-column table [S,P,O](Virtuoso uses it), and (2) grouping triples by predicate and defining a specific 2-column table: [S,O] for each one; this last technique, called vertical partitioning [1], is based on the fact that few predicates are used to describe a dataset. A third hybrid policy (implemented in systems like Jena-TDB) results from the combination of the previous ones: a specific 3-column table is considered to store clusters of correlated predicates. Hexastore [56] and RDF-3X [45] are well-known systems which create indexes for all ordering combinations (SPO, SOP, PSO, POS, OPS, OSP). Although their main goal is to achieve a global competitive performance, this index replication largely increases spatial requirements avoiding indexes to be fully loaded and queried in main memory. Thus, very expensive disk operations should be performed, resulting in less competitive operations. RDF-3X tries to reduce this effect by tuning a gap-based schema which compresses the leafs of the B+-tree storing all triples in the index. BitMat [7], suggests a compressed bit-matrix structure for storing huge RDF graphs. It represents the RDF data with a set of two-dimensional matrices: SO and OS for each predicate, PO for each subject and PS for each object (note that OP and SP are discarded). These matrices are gap-compressed by taking advantage of their sparseness. It also supports basic querying capability without decompressing the data. However, the vast majority of approaches suffer from lack of scalability [51], and use naive compression techniques. The work K^2-triples [5] constructs an RDF index over a compact data structure called k^2-trees [16], which excels at compressing very sparse two-dimensional binary matrices. K^2-triples applies vertical partitioning and constructs an independent k^2-tree for each predicate, which in turn stores all pairs (subject, object). The resulting k^2-trees describe very sparse 1 distributions which allow k^2-triples to achieve ultra-compressed representations.

7. Conclusions and Future Work

RDF publication and exchange at large scale are compromised by the textual representation formats used to encode huge datasets. They are very verbose and space-inefficient, they are not prepared to separate the metadata from the data itself, and more importantly, they require a full scan over the whole document to locate any piece of information. Therefore, we need more efficient formats that carry out this limitation.

We first propose a set of metrics and an empirical analysis of RDF Data to understand the structural features of RDF. This supports the decisions taken when proposing RDF data formats, data structures and indexes. An important finding is that the number of RDF terms that appear both as subject and object is very significative, but that does not occur for other combinations. This implies that compactness can be achieved by characterizing and grouping the references to the same node. Another interesting result is the analysis of degrees of the nodes, that provide insights not only for compression but also for designing indexes and query evaluation optimizers. We observe that objects are typically associated to only one predicate, whereas subjects are related to a set of limited predicates.

Based on the previous analysis, we devise `HDT`, a compact representation format to address the aforementioned problems by decomposing an RDF dataset into three logical components: Header, Dictionary, and Triples. This decomposition alone leads to space savings of up to 15 times compared to the original representation. Then, we build a specific compressor over `HDT`: `HDT-Compress`, which shows a size reduction to half of the achieved by traditional compressors. Our experiments demonstrate significant opportunities for RDF compression allowing important size reduction of the huge datasets that are being published in the Web of Data, therefore providing an efficient RDF exchange.

In addition, `HDT` opens new opportunities for practical processing of RDF: a carefully designed binary serialization does not only achieve compression, but also enables efficient retrieval mechanisms over the compressed representation. To this extent, `HDT` implements the idea that "*the data is the index*". The Triples component is specifically encoded using a succinct data structure that enables indexed access to any triple in the dataset. Thus, `HDT` provides effective RDF decomposition, simple compression notions and basic indexed access in a compact serialization format which provides efficient access to the data. In this sense, we have set up a website, `http://rdfhdt.org`, exposing HDT-based applications (such as HDT-it for visualization) and HDT compliant libraries (C++ and Java). Whereas the current approach already addresses the publication and exchange steps of the process, consumption of RDF data in `HDT` format deserves more research, particularly in efficient RDF indexing and SPARQL resolution. On the one hand, `HDT` describes a machine-friendly representation which makes any kind of post-processing much simpler. On the other hand, the `Check&Find` mechanism sets the basis for efficient retrieval. Currently, it is focused on single triple pattern retrieval, but full SPARQL resolution needs a comprehensive design of join operators and query optimization.

Acknowledgments

This work was partially funded by MICINN (TIN2009-14009-C02-02); Science Foundation Ireland (SFI): Grant No.∼SFI/08/CE/I1380, Lion-II; Fondecyt 1110287 and Fondecyt 1-110066. The first author is granted by Erasmus Mundus, the Regional Government of Castilla y León (Spain) and the European Social Fund. The second author is granted by the University of Valladolid: programme of Mobility Grants for Researchers (2012).

References

[1] D. Abadi, M. Adam, S.R. Madden, and K. Hollenbach. Scalable Semantic Web Data Management Using Vertical Partitioning. In *VLDB 2007*, pages 411–422, 2007.

[2] K. Alexander. RDF in JSON: A Specification for serialising RDF in JSON. In *SFSW 2008*, 2008.

[3] K Alexander, R Cyganiak, M Hausenblas, and J Zhao. Describing Linked Datasets-On the Design and Usage of voiD, the'Vocabulary of Interlinked Datasets'. In *LDOW 2009 at WWW 2009*, 2009.

[4] K. Alexander, R. Cyganiak, M. Hausenblas, and J. Zhao. Describing Linked Datasets with the VoID Vocabulary. http://www.w3.org/TR/void/, 2011. W3C Interest Group Note 03 March 2011.

[5] S. Álvarez, N. Brisaboa, J.D. Fernández, and M.A. Martínez-Prieto. Compressed k2-Triples for Full-In-Memory RDF Engines. In *AMCIS 2011, article 350*, 2011.

[6] M. Arias, J.D. Fernández, and M.A. Martínez-Prieto. An empirical study of real-world SPARQL queries. In *Proc. 1st International Workshop on Usage Analysis and the Web of Data (USEWOD)*, 2011.

[7] M. Atre, V. Chaoji, M.J. Zaki, and J.A. Hendler. Matrix "Bit" loaded: a scalable lightweight join query processor for RDF data. In *WWW 2010*, pages 41–50. ACM, 2010.

[8] D. Beckett. RDF/XML Syntax Specification (Revised). http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/, 2004. W3C Recommendation.

[9] D. Beckett and T. Berners-Lee. Turtle - Terse RDF Triple Language. http://www.w3.org/TR/2012/WD-turtle-20120710/, 2012. W3C Working Draft 10 July 2012.

[10] T. Berners-Lee. Notation 3, 1998. Available at http://www.w3.org/DesignIssues/Notation3.

[11] T. Berners-Lee. Linked Data: Design Issues. http://www.w3.org/DesignIssues/LinkedData.html, 2006. Retrieved October 2012.

[12] C. Bizer, T. Heath, and T. Berners-Lee. Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems*, 5:1–22, 2009.

[13] C. Bizer, T. Heath, K. Idehen, and T. Berners-Lee. Linked Data On the Web (LDOW 2008). In *WWW 2008*, pages 1265–1266. ACM, 2008.

[14] P. Boldi and S. Vigna. The webgraph framework I: compression techniques. In *WWW 2004*, pages 595–602, 2004.

[15] D. Brickley. RDF Vocabulary Description Language 1.0: RDF Schema. http://www.w3.org/TR/rdf-schema/, 2004. W3C Recommendation.

[16] N. Brisaboa, S. Ladra, and G. Navarro. K^2-trees for Compact Web Graph Representation. In *SPIRE 2009*, LNCS 5721, pages 18–30. Springer, 2009.

[17] J. J. Carroll. Signing RDF Graphs. In *ISWC 2003*, pages 369–384, 2003.

[18] F. Chierichetti, R. Kumar, and P. Raghavan. Compressed web indexes. In *WWW 2009*, pages 451–460, 2009.

[19] E.I. Chong, S. Das, G. Eadon, and J. Srinivasan. An efficient sql-based rdf querying scheme. In *VLDB 2005*, pages 1216–1227, 2005.

[20] D. Clark. *Compact PAT trees*. PhD thesis, University of Waterloo, 1996.

[21] J.G. Cleary and I.H. Witten. Data Compression Using Adaptive Coding and Partial String Matching. *IEEE Transactions on Communications*, 32(4):396–402, April 1984.

[22] R. Cyganiak, S. Field, A. Gregory, W. Halb, and J. Tennison. Semantic statistics: Bringing together SDMX and SCOVO. *LDOW 2010 at WWW 2010*, pages 2–6, 2010.

[23] R. Cyganiak, A. Harth, and A. Hogan. N-Quads: Extending N-Triples with Context, 2008. Available at http://sw.deri.org/2008/07/n-quads/. Retrieved October 2012.

[24] R Cyganiak, H Stenzhorn, R Delbru, S Decker, and G Tummarello. Semantic sitemaps: Efficient and flexible access to datasets on the semantic web. In *ESWC 2008*, volume 5021, pages 690–704. Springer-Verlag, 2008.

[25] L. Ding and T. Finin. Characterizing the Semantic Web on the Web. In *ISWC 2006*, pages 242–257, 2006.

[26] O. Erling and I. Mikhailov. RDF Support in the Virtuoso DBMS. *Proceedings of CSSW*, 221:59–68, 2007.

[27] J.D. Fernández, C. Gutierrez, and M.A. Martínez-Prieto. RDF compression: basic approaches. In *WWW 2010*, pages 1091–1092, 2010.

[28] R. González, S. Grabowski, V. Makinen, and G. Navarro. Practical implementation of rank and select queries. In *WEA 2005*, pages 27–38, 2005.

[29] C. Gutierrez, C. Hurtado, A.O. Mendelzon, and J. Perez. Foundations of semantic web databases. *Journal of Computer and System Sciences (JCSS)*, 77:520–541, 2011.

[30] J. Hayes and C. Gutierrez. Bipartite Graphs as Intermediate Model for RDF. In *ISWC 2004*, pages 47–61, 2004.

[31] A. Hogan. *Exploiting RDFS and OWL for Integrating Heterogeneous, Large-Scale, Linked Data Corpora*. PhD thesis, DERI, 2011.

[32] A. Hogan, A. Harth, A. Passant, S. Decker, and A. Polleres. Weaving the pedantic web. In *LDOW 2010 at WWW 2010*, Raleigh, USA, April 2010.

[33] Aidan Hogan, Axel Polleres, Jürgen Umbrich, and Antoine Zimmermann. Some entities are more equal than others: statistical methods to consolidate Linked Data. In *Workshop on New Forms of Reasoning for the Semantic Web: Scalable & Dynamic (NeFoRS2010)*, 2010.

[34] Aidan Hogan, Antoine Zimmermann, Jürgen Umbrich, Axel Polleres, and Stefan Decker. Scalable and distributed methods for entity matching, consolidation and disambiguation over linked data corpora. *Web Semantics: Science, Services and Agents on the World Wide Web*, 10:76 – 110, 2012.

[35] D.A. Huffman. A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[36] IBM. *IBM Dictionary of Computing*. McGraw-Hill, 1993.

[37] A. Langegger and W. Woss. RDFStats - An Extensible RDF Statistics Generator and Library. In *DEXA 2009*, pages 79–83, 2009.

[38] R. R.: Lassila O., Swick. Resource description framework (rdf) model and syntax specification. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/, 1999.

[39] D. Le-Phuoc, J. X. Parreira, V. Reynolds, and M. Hauswirth. RDF On the Go : An RDF Storage and Query Processor for Mobile Devices. In *ISWC 2010*, 2010. Available at http://iswc2010.semanticweb.org/pdf/503.pdf.

[40] M.A. Martínez-Prieto, J.D. Fernández, and R. Cánovas. Querying RDF Dictionaries in Compressed Space. *ACM SIGAPP Applied Computing Reviews*, 12(2):64–77, 2012.

[41] D. McGuinness and F. van Harmelen. OWL Web Ontology Language Overview. http://www.w3.org/TR/owl-features/, 2004. W3C Recommendation.

[42] B. Motik, P. F. Patel-Schneider, and B. Parsia. OWL 2 Web Ontology Language Structural Spcification and Functional Style-Syntax. http://www.w3.org/TR/owl2-syntax/, 2009. W3C Recommendation.

[43] T. Neumann and G. Weikum. RDF-3X: a RISC-style engine for RDF. *Proceedings of the VLDB Endowment*, 1(1):647–659, 2008.

[44] T. Neumann and G. Weikum. Scalable join processing on very large rdf graphs. In *COMAD 2009*, pages 627–640, 2009.

[45] T. Neumann and G. Weikum. The RDF-3X engine for scalable management of RDF data. *The VLDB Journal*, 19(1):91–113, 2010.

[46] E. Oren and et al. Sindice.com: a document-oriented lookup index for open linked data. *International Journal of Metadata, Semantics and Ontologies*, 3(1):37–52, 2008.

[47] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and Complexity of SPARQL. *ACM Transactions on Database Systems*, 34(3):1–45, 2009.

[48] E. Prud'hommeaux. SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/, 2008. W3C Recommendation.

[49] Michael Schmidt, Thomas Hornung, Norbert Küchlin, Georg Lausen, and Christoph Pinkel. An Experimental Comparison of RDF Data Management Approaches in a SPARQL Benchmark Scenario. In *ISWC 2008*, pages 82–97, 2008.

[50] J. Schneider and T. Kamiya. Efficient XML Interchange (EXI) Format 1.0, 2009. W3C Candidate Recommendation.

[51] L. Sidirourgos, R. Goncalves, M. Kersten, N. Nes, and S. Manegold. Column-store support for RDF data management: not all swans are white. *VLDB*, 1(2):1553–1563, 2008.

[52] M. Stonebraker, D.J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, and Others. C-store: a column-oriented DBMS. In *Proceedings of the 31st*

international conference on Very large data bases, pages 553–564. VLDB Endowment, 2005.

[53] Y. Theoharis, Y. Tzitzikas, D. Kotzinos, and V. Christophides. On Graph Features of Semantic Web Schemas. *IEEE Trans. on Know. and Data Engineering*, 20(5):692–702, 2008.

[54] J. Urbani, J. Maassen, and H. Bal. Massive semantic web data compression with mapreduce. In *HPDC 2010*, pages 795–802. ACM, 2010.

[55] J. Weaver and G.T. Williams. Reducing I/O Load in Parallel RDF Systems via Data Compression. In *1st Workshop on High-Performance Computing for the Semantic Web (HPCSW 2011)*, 2011.

[56] C. Weiss, P. Karras, and A. Bernstein. Hexastore: sextuple indexing for semantic web data management. *Proceedings of the VLDB Endowment*, 1(1):1008–1019, 2008.

[57] K Wilkinson, C Sayers, H Kuno, and D Reynolds. Efficient RDF Storage and Retrieval in Jena2. *Proceedings of SWDB*, 3(September):7–8, 2003.

Appendix A. Experimental RDF Characterization

This appendix comprises an experimental study on real-world datasets in order to characterize RDF structure and redundancy by applying the parameters presented in Section 2.1. We chose five datasets based on the huge amount of triples, different application domains and previous uses in benchmarking. Table A.3 summarizes the most basic features of the datasets. *Geonames*, *Dbtune* and *Uniprot* are samplings extracted from the Billion Triples Challenge 2010 data collection[18]. In particular, *Geonames* gathers geographic concepts, *Dbtune* holds music information and *Uniprot* is a freely-accessible RDF dataset of protein sequence data. *Wikipedia³* [19] stands for the English Wikipedia links between pages transformed to RDF. *DBpedia*[20] is an RDF conversion of Wikipedia, with the aim of making this type of information semantically available on the Web.

The column "Size" in Table A.3 shows the original size in raw N-Triples format, "Triples" indicates the number of triples in the dataset and the three latest columns show the number of different subjects, predicates and objects respectively. As expected, the number of predicates stays commonly low. We have also chosen *Dbpedia* as an extreme case in which the number of predicates grows to the order of thousands. However, note that they remain proportionally small to the number of triples. Thus, the consideration of DBpedia broadens the experimentation of common datasets which use more limited-size predicate dictionaries.

We compute the parameters previously presented, in order to characterize the structure and gain insights toward analyzing the redundancy of each dataset, as well as their compact and compression possibilities.

[18]http://km.aifb.kit.edu/projects/btc-2010/

[19]http://labs.systemone.at

[20]DBpedia dump: http://wiki.dbpedia.org/Downloads36

A preprocessing step is firstly applied. Billion Triples data was parsed from N-Quads format[21] to N-Triples by eliminating context information, gathering the selected datasets. Duplicate triples were discarded (Table A.3 reflects the number of triples after cleaning).

Table A.16 summarizes the data statistics collected for the different datasets. Note that these datasets are ordered in increasing number of triples. Several comments are in order: first of all, we remark the high variability of values among the datasets. Although expected, this points out the problem of designing a general "one size fits all" solution for RDF engines. Moreover, these metrics encourage the study of each particular case as well as the interest in getting insights on common behaviors.

Max out-degree reveals that there are some subjects present in many triples (*e.g.* 7408 in Wikipedia³) but the mean out-degree remains lower. This corresponds to the presence of a subject power-law distribution. For instance, the subjects of *Uniprot* occur in a mean of only 5.94 triples and those of the largest dataset (*Dbpedia*) occur in a mean of 12.62 triples. However, this increase cannot be attributed to the size of the datasets but the nature of them. *DBpedia* describes facts of things, so more statements about a subject have to be included. In contrast, the number of links between proteins in *Uniprot* is limited by the underneath biological theory.

If we compare the out-degree with the partial, labeled and direct degrees, the structure around subjects is further detailed. First of all, partial and direct out-degree have similar figures. Thus, the maximum out-degrees are given by one (s, p) pair with multiple related objects. In contrast, mean partial out-degree is slightly bigger than 1, which implies that the presence of multivalued pairs (s, p) is not so frequent.

The labeled out-degree verifies that few predicates are related to the same subject. For instance, although *Dbpedia* has 39,672 predicates, at most 22 links the same subject. In all cases, the mean labeled out-degree ranges between 4 and 5, which is a clear indicator of the presence of star-shaped nodes, *i.e.*, nodes with different triples around one common subject.

The study of the in-degree comes to very similar conclusions. In addition, maximum degrees reveal a more skewed structure on objects, *i.e.*, clearer power-law distributions are present. An interesting conclusion emerges when studying mean labeled in-degree. The number of predicates related to a given object is very close to 1. This stands for a specific treatment of these "leave nodes" for each predicate. Thus, approaches such as a specific compression over vertical partitioning (such as C-Store), can obtain important results.

[21]http://sw.deri.org/2008/07/n-quads/

dataset	Size (GB)	# Triples	# Subjects	# predicates	# Objects
Geonames	1.00	9,415,253	2,203,561	20	3,031,664
Wikipedia^3	6.72	47,054,407	2,162,189	9	8,268,864
Dbtune	9.34	58,920,361	12,401,228	394	14,264,221
Uniprot	9.11	72,460,981	12,188,927	126	9,084,674
Dbpedia-en	33.12	232,542,405	18,425,128	39,672	65,200,769

Table A.3: datasets description.

				Geonames	Wikipedia^3	Dbtune	Uniprot	Dbpedia-en
SUBJECT OUT-DEGREE	*Max*	*total*	$deg^-(G)$	369.00	7408.00	2194.00	2408.00	7184.00
		partial	$deg^{--}(G)$	298.00	7400.00	2193.00	2406.00	7177.00
		labeled	$degL^-(G)$	17.00	7.00	24.00	22.00	448.00
		direct	$degD^-(G)$	369.00	7408.00	2194.00	2408.00	7184.00
	Mean	*total*	$\overline{deg^-}(G)$	4.27	21.76	4.75	5.94	12.62
		partial	$\overline{deg^{--}}(G)$	1.07	3.95	1.14	1.30	2.06
		labeled	$\overline{degL^-}(G)$	4.00	5.51	4.16	4.59	6.13
		direct	$\overline{degD^-}(G)$	4.27	21.75	4.68	5.93	11.17
OBJECT IN-DEGREE	*Max*	*total*	$deg^+(G)$	2.20×10^6	2.05×10^6	2.27×10^6	6.04×10^6	7.33×10^6
		partial	$deg^{++}(G)$	2.20×10^6	2.05×10^6	2.27×10^6	6.04×10^6	7.33×10^6
		labeled	$degL^+(G)$	2.00	4.00	93.00	15.00	2938.00
		direct	$degD^+(G)$	2.20×10^6	2.05×10^6	2.27×10^6	6.04×10^6	2080.00
	Mean	*total*	$\overline{deg^+}(G)$	3.11	5.69	4.13	7.98	3.57
		partial	$\overline{deg^{++}}$	3.08	5.40	3.87	5.75	2.72
		labeled	$\overline{degL^+}$	1.00	1.05	1.07	1.39	1.31
		direct	$\overline{degD^+}$	3.11	5.69	4.07	7.95	55.03
PREDICATE DEGREE	*Max*	*total*	$degP(G)$	2.35×10^6	3.45×10^7	1.23×10^7	1.43×10^7	9.87×10^7
		out	$degP^-(G)$	1.65×10^6	3.84×10^6	2.25×10^6	2.01×10^7	1.14×10^7
		in	$degP^+(G)$	2.20×10^6	2.16×10^6	1.00×10^7	1.21×10^7	8.89×10^6
	Mean	*total*	$\overline{degP}(G)$	470763.00	5.22×10^6	149544.00	575087.00	5861.63
		out	$\overline{degP^-}(G)$	152709.00	967404.00	38641.10	100034.00	2158.84
		in	$\overline{degP^+}$	440438	1.32×10^6	130964.00	443981.00	2845.39
RATIOS			α_{s-o}	0.018	0.17	0.61	0.43	0.25
			α_{s-p}	0	0	0	0	5.76×10^{-4}
			α_{p-o}	0	0	3.44×10^{-6}	1.75×10^{-6}	0

Figure A.16: datasets statistic summary.

A remarkable result is the great difference between the subject and object distribution in *Dbpedia*. Objects are related with subjects 5 times more than vice versa (55.03 versus 11.17).

Predicate degrees impact on vertical partitioning, hence the relevance of their study. Except for *Dbpedia*, the difference between the maximums and means predicate degrees are not as clear as the common out and in-degrees. This stands for a distribution out of the scope of power law, although some predicates (such as `rdf:type`) could be predominant over the rest. In fact, the distribution of predicates is clearly determined by the design and purpose of the dataset. For instance, every protein in *Uniprot* is characterized with the same number of properties, only diverging in the number of links. The same case might be present in categorizing an artist or a song in *Dbtune*. However, the properties over a resource can vary greatly in *Dbpedia*.

Finally, the ratios reveal a level of cohesion between the different types of nodes. As we expected, subject-predicate and predicate-object ratios are almost negligible. These are scheme descriptions, which are rare due to the RDF itself is schema-relaxed, *i.e.*, the vocabulary evolves as needed on demand. Subject-object is the most frequent path constructor. For instance, the design of *Dbtune* and *Uniprot* has cohesion (61% and 43% respectively are shared subjects-objects), with a high subject-object ratio and a smaller number of very frequent predicates. This reveals a star chained design in which a subject is strongly characterized and interlinked with others. A similar interpretation could be done for *Wikipedia^3* and *Dbpedia* but with less cohesion. Note that one could think that *Wikipedia^3* or *Dbpedia* pages have links to other pages. However, the number of properties over a page or resource is proportionally higher than the number of links to other pages. In turn, the low subject-object ratio in *Geonames* shows that data are descriptions of geographical concepts which, at most, are linked to their superior administrative category.

Regarding power-law distributions, Figure A.17 shows the distribution of subjects and objects of *Wikipedia^3*. As we expected, a power-law distribution

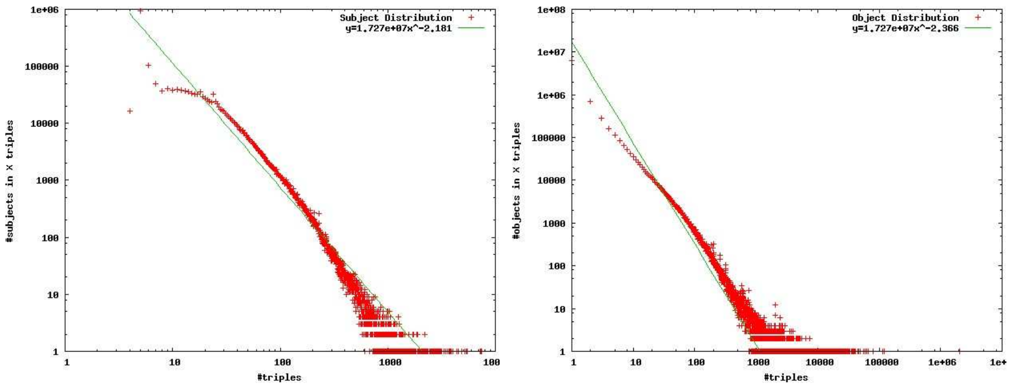


Figure A.17: Wikipedia³ distribution of subjects (left) and objects (right), *e.g.* a point (X,Y) in the rightmost graphic says that there are Y different objects each occurring in X triples. Both axis are logarithmic. The power laws have exponent -2.181 and -2.366 respectively.

is remarkable in both cases. The other datasets reveal the same distribution for subject and object.

All these results immediately point to possible compact design models of RDF and optimizations of indexing techniques. Our approach, HDT, exploits the significant correlation and the inherent redundancy in data and structure. In particular, the deployment proposed in Section 4 takes advantage of subject-object ratio characterization and groups the references to the same node in the Dictionary. In turn, the proposed Triples component represents the graph compacting the distribution with implicit and coordinated adjacency lists, parametrized by the degree metrics.

Appendix B. Binary RDF on HDT

HDT can be further developed into a complete RDF syntax toward a binary representation for RDF. Let us define some basic concepts previous to the HDT syntax specification described in this appendix.

Definition 4 (HDT processor)**.** *A program module called an* **HDT processor***, whether it is software or hardware, is used by application programs to encode their data into* **HDT core data** *and/or to decode* **HDT core data** *to make the data accessible. The former and latter aforementioned roles of* HDT *processors are called* **HDT encoder** *and* **HDT decoder** *respectfully.*

Definition 5 (HDT core data)**. HDT core data** *consists of the Dictionary and Triples information of the* HDT *representation, whether it is present in a unique or several files or streams. This core data must be self-contained,* i.e.*, it must contain enough information to consume the dataset. Each file or stream belonging to the* HDT *core data is headed by control information and followed by the* **HDT body** *which can be the full or part of the Dictionary component or the Triples component or both. Note that the Header is out of the definition of the* HDT *core data, constituting a different file or stream.*

Figure A.18 shows a typical producer/consumer case in HDT. First, the producer uses an HDT encoder in order to generate HDT from RDF. The Header, if present, can be retrieved by the consumer in order to get metadata about the dataset and the publication. In turn, the consumer uses an HDT decoder to efficiently access the HDT core data. Furthermore, the HDT decoder should provide the consumer with distinct access possibilities, such as getting the original full RDF document, querying over the data or several management operations.

Appendix B.1. HDT Syntax

The syntax of HDT is given by the syntax of the Header, and the Dictionary and the Triples encodings.

Appendix B.1.1. Header

The Header has been described as a flexible component containing metadata about the data publication together with information to retrieve and process the HDT core data. The desired operations over the Header and its requirements of machine-readable, human-friendly, and easy querying, are well satisfied considering the Header as an RDF graph. This allows expressing metadata about the dataset (originally in RDF) with an RDF syntax, which can be discovered and used through well-known mechanisms, such as SPARQL Endpoints.

Note that the Header is out of the scope of the HDT core data and can be accessed independently.

The use of VoiD [3] as the main vocabulary of the Header is strongly recommended. The Header should be represented in any RDF syntax. The normative format of the Header is RDF/XML.

Appendix B.1.2. Control Information

Dictionary and Triples components could be distributed or chunked, hence each part should have enough information for processing it properly. Thus, each Dictionary and Triples component, as well as subparts of

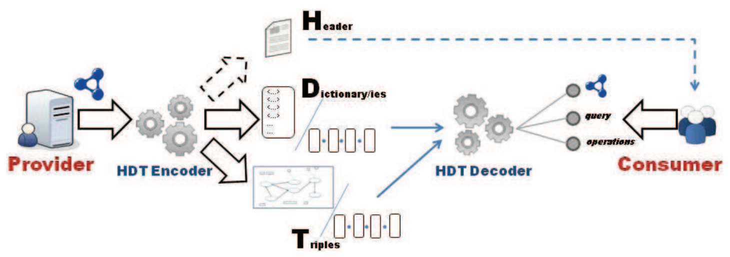


Figure A.18: The process of HDT encoding/decoding.

Cookie	Format Version	Component Bits	Presence Bit for Options	[Options]

Table B.4: Control Information.

HDT Format Version	Stands for
0000	Version 1
0111	Version 8
1000 0000	Version 9

Table B.5: Version Examples.

them in case of splitting, is headed by a sequence representing control information.

The control information can identify Dictionary and Triples components or subparts of them, distinguish HDT core data streams from text, identify the version of the HDT format being used, and specify the options used to process the HDT core data. The control information has the structure shown in Table B.4.

The control information starts with an HDT Cookie, a four byte field that serves to indicate that the file or stream of which it is a part is HDT core data. The four byte field consists of four characters “ $ ” , “ H ”, “ D ” and “ T ” in that order, each represented as an ASCII octet. This four byte sequence is particular to HDT and specific enough to distinguish HDT files and streams from a broad range of data types.

The second part of the control information identifies the version of the HDT format being used. The version is a sequence of one or more 4-bit unsigned integers. The first bit set to 1 indicates that the next 4-bits must be read. The version number is determined by summing each sequence of 4-bit unsigned integers and adding 1 (one). Table B.5 show examples of versions.

The third part of the control information consists in the HDT component distinguishing bits, which identify the component or components of the HDT body that follow the control information. The HDT component distinguishing bits are three bits in which the combination of the first two bits identifies the information below the HDT control information, as shown in Table B.6. The third bit set to ’1’ indicates that the current file or stream is a subpart of the entire component. When both Dictionary and Triples Component, *i.e.* HDT core data follow

Component Bits	Stands for
01	Dictionary Component
10	Triples Component
11	both Dictionary and Triples Component
00	Reserved

Table B.6: 1st and 2nd HDT component distinguishing bits.

Property	Use
codification	Identify the codification scheme of the HDT body.
format	Set up the MIME type of the HDT body.
[user defined]	User defined metadata.

Table B.7: HDT Options.

the control information.

The HDT options field provides a mechanism to specify the options of the encoded component or components of the HDT body. This field is optional and its presence is indicated by the value of the presence bit that follows the HDT component distinguishing bits. If the HDT options are present (presence bit sets to ’1’), then a final reserved word “$END” must be added at the end of the control information to delimit its length. When HDT options are present, an HDT Processor must observe the specified options to process the HDT body. Otherwise, an HDT Processor may follow the default values. If the Header component is present and it informs about such information, their values override the default ones. In case of conflicts, the HDT options override the information of the Header.

HDT Options are represented as text with a <property>:<value>; scheme, as shown in Table B.7.

The codification is used to identify the concrete codification scheme used to process the HDT body. This must be an URI identifying the codification of the component or components indicated by the HDT component distinguishing bits. When the codification option is absent, undefined or empty, no statement is made about the codification scheme, therefore an HDT Processor should assume the codification by default of the component or components indicated by the HDT component distinguishing bits, unless other information is communicated out of band. Table B.8 shows the URIs of the codifications by default and the section with detailed information.

HDT Component	Codification by default	Section
Dictionary	hdt:dictionaryPlain	Section 4.2
Triples	hdt:triplesBitmap	Section 4.3.3
both D. and T.	hdt:globalBitmap	Section 4.3.3

Table B.8: HDT default URI codifications.

HDT Component	Format by default
Dictionary	text/plain
Triples	application/octetstream
both D. and T.	multipart/mixed

Table B.9: HDT default formats.

The format sets up the MIME type of the HDT body. If the format option is absent, undefined or empty, an HDT Processor should assume the format by default of the component(s) indicated by the HDT component distinguishing bits, unless other information is communicated out of band. Table B.9 shows the formats by default.

The user defined metadata sets up auxiliary properties with additional control information to process the data. The syntax and semantic of the user defined properties depend on the codification of the HDT body, thus HDT Processor must interpret the user defined properties within the codification context. User defined properties must follow the following naming conventions to prevent conflicts between different component properties:

- Dictionary Component. Property names must start with one unique dollar sign("$") character followed by alphanumeric characters.
- Triples Component. Property names must start with two unique dollar signs ("$$") characters followed by alphanumeric characters.
- Both Dictionary and Triples Component. Property names must start with three unique dollar signs("$$$") followed by alphanumeric characters.

Appendix B.1.3. Dictionary

The main goal of the Dictionary is to assign a unique ID to each element in the dataset and thus there is no restriction on the particular mapping or codification. We provide a general dictionary encoding to be followed by every particular codification. A dictionary codification by default is given, and it must be assumed by HDT Processors when no other codification scheme is specified in the codification option or in the Header component.

Dictionary Encoding. The distinction between URIs, literals and blanks, as well as string escaping, follows a similar N3 syntax:

- URIs are delimited by angle brackets "<" and ">". Whitespace within them is to be ignored.

	Stands for
a	`<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>`
=	`<http://www.w3.org/2002/07/owl#sameAs>`
=>	`<http://www.w3.org/2000/10/swap/log#implies>`
<=	`<http://www.w3.org/2000/10/swap/log#implies>, but in the inverse direction`

Table B.10: Dictionary predefined prefixes.

- URIs can be absolute or relative to the base URI (defined in the user defined metadata or in the Header component).
- URIs can make use of prefixes (defined in the user defined metadata or in the Header component) or predefined prefixes (described below). Blanks are named with the _: namespace prefix, *e.g.* _:b19 represents a blank node.
- Literals are written using double-quotes (*e.g.* "literal"). The ""literal"" string form is used when they may contain linebreaks.
- Literals represented numbers or booleans can be written directly corresponding to the right XML Schema Datatype: xsd:integer, xsd:double, xsd:decimal or xsd:boolean.
- Comments are not allowed in any form.

Table B.10 shows the predefined prefixes whereas the string escaping sequences follows strictly N3.

Appendix B.1.4. Triples

The Triples component must contains the structure of the data after the ID replacement, comprising the pure structure of the underlying graph. The multiple indexing variants, uses and allowed operations over the triples difficult the restriction on a particular codification for the Triples component. Instead, any triple codification may be established by specifying a concrete codification option or by the Header component.

The codification by default must be interpreted as Bitmap Triples (Section 4.3.3) by HDT Processors.

When the same file or stream comprises both the Dictionary component and the Triples component, then a secondary control information must be included in between the two streams, identifying the HDT component distinguishing bits of each stream.