# Compact Representation of Large RDF Data Sets for Publishing and Exchange ⋆

Javier D. Fernández[1], Miguel A. Martínez-Prieto[1,2], and Claudio Gutierrez[2]

[1] Department of Computer Science, Universidad de Valladolid (Spain)
{jfergar,migumar2}@infor.uva.es
[2] Department of Computer Science, Universidad de Chile (Chile)
cgutierr@dcc.uchile.cl

**Abstract.** Increasingly huge RDF data sets are being published on the Web. Currently, they use different syntaxes of RDF, contain high levels of redundancy and have a plain indivisible structure. All this leads to fuzzy publications, inefficient management, complex processing and lack of scalability. This paper presents a novel RDF representation (HDT) which takes advantage of the structural properties of RDF graphs for splitting and representing, efficiently, three components of RDF data: *Header*, *Dictionary* and *Triples* structure. On-demand management operations can be implemented on top of HDT representation. Experiments show that data sets can be compacted in HDT by more than fifteen times the current naive representation, improving parsing and processing while keeping a consistent publication scheme. For exchanging, specific compression techniques over HDT improve current compression solutions.

## 1 Introduction and Related Work

The intended goal of the original RDF/XML representation design was to add small descriptions (metadata) to documents, to protocols, to mark web pages or to describe services. Representations like N3, Turtle and RDF/JSON, although having improved, in several respects, the original format, are still dominated by a *document-centric* view. Today, when one of the major trends in the development of the Web is RDF publishing at large scale, *i.e.* make RDF data publicly available for unknown purposes and users, the need to consider RDF under a *data-centric* view is becoming indispensable.

An analysis of published RDF data sets (the 2000 US Census, DBpedia, GeoNames, Uniprot, DBLP, etc.) reveals several undesirable features. First, the provenance and metadata about contents are barely present, and their information is neither complete nor systematic. Second, the files have neither internal structure nor a summary of their content. Basic data operations have to deal with the sequentiality of the information in the file, thus parsing the whole data and in most cases including human operation because the metadata is outside the file. For mashups of different sources, the situation is worse. Currently, the effort to prepare the data to be published is so costly, that

⋆ Partially funded by MICINN (TIN2009-14009-C02-02), Millennium Institute for Cell Dynamics and Biotechnology (ICDB) (Grant ICM P05-001-F), and Fondecyt 1090565. The first author is granted by a fellowship from Erasmus Mundus, the Regional Government of Castilla y Leon (Spain) and the European Social Fund. Special thanks to M. Gagie.

files commonly have no design, no plan and no user in mind. They resemble unwanted creatures whose owners are keen to be rid of them.

This state of affairs does not scale to a Web where large data sets will soon, increasingly, be produced dynamically and automatically. Furthermore, most data would have to be *machine-understandable* in line with the aim of the original Semantic Web project. Thus, scaling the process of publishing and exchanging large RDF data sets should comply with some basic features. At the *logical level*, a large-scale data set should have standard metadata, like provenance (source, providers, publication date, etc.), editorial metadata (publisher, date, version, etc.), data set statistics (size, quality, type of data, basic parameters of the data) and intellectual property (types of copy[left|right]s). At the *physical level*, RDF representation at large scale should permit efficient processing, managing and exchanging (between systems and memory-disk movements). At the format level, desirable features include simple checks for triple existence, redundancy minimization and modular construction. Imagine a user who wants to publish or exchange a large data set from her preferred RDF data store. She would first need to dump the data into one RDF format, and then, due to the large size of the data, possibly compress it with a generic compressor. The resultant file has no structure, no metadata and it is hardly usable natively, *i.e.* without an appropriate external tool (another RDF data store, a visualization software, etc.).

This paper addresses these challenges, and shows that there are feasible and simple solutions. In particular, we introduce a new representation format (*Header-Dictionary-Triples*: `HDT`) that modularizes the data and uses the skewed structure of big RDF graphs [10, 19, 21] to achieve large spatial savings. It is based on three main components:

- A *header*, including logical and physical metadata describing the RDF data set. This serves as an entrance point to the information on the data set.
- A *dictionary*, organizing all the identifiers in the RDF graph.This provides a catalog of the information entities in the RDF graph with high levels of compression.
- A set of *triples*, which comprises the pure structure of the underlying RDF graph while avoiding the noise produced by long labels and repetitions.

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

Figure 1 shows a step-by-step description of the process to obtain an `HDT` representation of an RDF graph. 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 have the `HDT` concrete implementation for publication and exchange of RDF.

If we go back to the example of the user who wants to publish or exchange large RDF data, the advantages of `HDT` can be summarized as follows: 1) More compact and compressible: uses much less space, thus saving storing space and communication bandwidth and time; 2) Is clean and modular: it separates dictionary from triples (the graph structure), includes a header with metadata about the data; 3) Permits basic data operations by allowing access to parts of the graph without needing to process all of it.

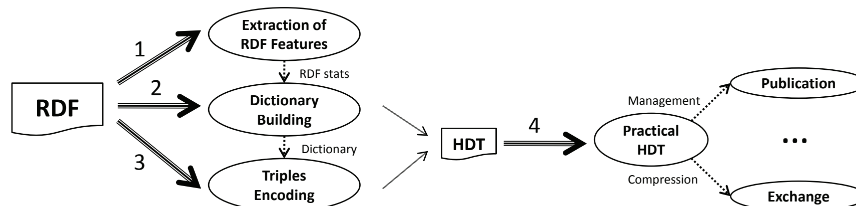



**Fig. 1.** A Step-by-step construction of the HDT format from a set of triples

The paper is organized as follows. Section 2 starts defining the set of metrics to characterize the structural RDF features used in HDT. Next, the HDT format is presented by an individual description of each component: Header, Dictionary, and Triples. Section 3, firstly details the practical implementation approach for HDT. Then, we detail the HDT management and compression. This section ends with an empirical study which analyzes the current HDT features on real-world data sets. Section 4 gives a brief discussion and addresses some future work. Finally, the appendix provides a study of the structural features of the data sets used in our experimentation, analyzing their impact on HDT. Additional resources and examples are available at `http://hdt.dcc.uchile.cl`.

*Related Work.* Today there are several representations for RDF data, *e.g.* RDF/XML [3], N3[3],Turtle[4], and RDF/JSON [1]. None of these proposals, though, seems to have considered data volume as a primary goal. RDF/XML, due to its verbosity is good for exchanging data, but only on a small scale. Turtle (a sub-language of N3) is a more compact and readable alternative. Although these formats present features to "abbreviate" constructions like URIs, groups of triples, common datatypes or RDF collections, the compactness of the representation definitively was not the main concern of their design. RDF/JSON resembles Turtle, with the advantage of being coded in a language easier to parse and more widely accepted in the programming world.

Regarding the structure of RDF real-world data, several studies point to the presence of power-law distribution, in term frequencies [10], resources [19] and schemas [21].

RDF compression capabilities have been studied [11] 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 results have been applied in the design of RDF data sets. There is little work on the design of large RDF data sets. There have been projects discussing design issues of RDF[5], and a working group on design issues of translation from relational databases to RDF[6]. However, none of these works have touched the problem of RDF publication

[3] http://www.w3.org/DesignIssues/Notation3

[4] http://www.w3.org/TeamSubmission/turtle/

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

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

and exchange at large. The project that is currently systematically addressing the issue of publication of RDF at large, Linked Data, is starting to face some of these issues.

## 2 Compacting RDF with HDT: The Concepts

### 2.1 Taking advantage of the skewed structure of real-world RDF data

Although power-law[7] 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 [10] 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[5]), the space of instances is sparsely populated, since most classes and properties have never been instantiated. By crawling the Web, Oren[19] comes to similar conclusions, showing that resources (URIs) in different documents fit to a power-law distribution. Theoharis [21] studies these properties for Semantic Web schemas, RDFS and OWL[16]. 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) [17, 18]. This schema analysis has contributed to synthetic schema generation for benchmarking [21]. These results motivate the adaptation 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 [4, 6].

For our purposes, a few indicators of the graph structure will be sufficient. RDF graph notation will follow [13, 20], with no distinction between URIs, Blank nodes and Literals. A triple then, $(s, p, o)$, is represented as a labeled graph $s \xrightarrow{p} o$. 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 (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 of $G$ in which $s$ occurs as subject. Formally, $deg^-(s) = |\{(s, y, z)/(s, y, z) \in G\}|$. The* maximum out-degree*, $deg^-(G)$, and the* mean out-degree*, $\overline{deg^-}(G)$, are defined as the maximum and mean out-degrees of all subjects in $S_G$.*
2. *The* partial out-degree *of $s$ 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, $deg^{--}(s, p) = |\{(s, p, z)/(s, p, z) \in G\}|$. The* maximum partial out-degree *of $G$, $deg^{--}(G)$, and* mean partial out-degree*, $\overline{deg^{--}}(G)$, 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 $degL^-(s)$, 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, $degL^-(s) = |\{p/p \in P_G, (s, p, z) \in G\}|$. The* maximum labeled out-degree *of $G$, $degL^-(G)$, and* mean labeled out-degree*, $\overline{degL^-}(G)$, are defined as the maximum (resp. the mean) labeled out-degrees of all subjects of $G$.*

[7] 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 $a, c, \beta$ constants.

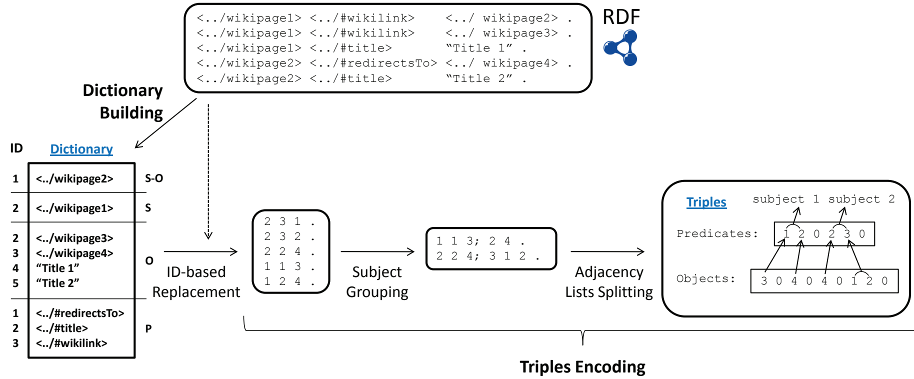



**Fig. 2.** Incremental representation of an RDF data set with `HDT`

Symmetrically, we define for objects the *in-degree*, denoted $deg^{+}(o)$, partial in-degree, $deg^{++}(o,p)$ and labeled in-degree, $degL^{+}(o)$. Their corresponding maximums and means are denoted as $deg^{+}(G)$, $deg^{++}(G)$, $degL^{+}(G)$, $\overline{deg^{+}}(G)$, $\overline{deg^{++}}(G)$ and $\overline{degL^{+}}(G)$. An additional property will be needed in what follows:

**Definition 2 (subject-object ratio $\alpha_{s-o}$).** *It is defined as the proportion of common subjects and objects in the graph G. Formally,* $\alpha_{s-o} = \frac{|S_G \cap O_G|}{|S_G \cup O_G|}$.

Out-degree indicates the relevance of a subject node. A node with high out-degree, also called star, will have hundreds, or even thousands, of arcs (labeled edges in RDF). In conjunction with maximum and mean values, this constitutes good evidence of the existence of these types of nodes in a given graph. Similar reasoning can be made for 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 out-degree provides a metric of the multi evaluation of pairs (subject-predicate or predicate-object), while labeled degree refines the star-nodes categorization. Finally, 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 which are described elsewhere, or as objects describing other resources. In the final Appendix we illustrate these parameters for three real-world data sets.

In what follows, we will use these characteristics to provide a compact structure that represents, succinctly, the information of an RDF data set. Figure 2 outlines the incremental processing of our proposal. The result splits the RDF data set into three components that are represented and managed efficiently.

### 2.2 Header

The Header component is responsible for providing metadata information about the RDF collection. Although the most used RDF syntaxes consider the possibility of including metadata information, they present several drawbacks. Metadata is provided in

the same RDF syntax as the data set, inheriting some of its problems and making difficult the automatic distinction between data and metadata. The metadata of the collection remains unclear and its management is inefficient.

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

- *Source and provider information*. This includes all kind of authority information about the source (or sources) of data and the provider of the data set, which can differ from the creator of data (*e.g.* in mashup applications). Note that this information can be shared between several data sets of a provider.
- *Publication data*. This collects the metadata about the publication act, such as the site of publication, dates of creation and modification, version of the data set (which could be useful for updating notification), language, encoding, namespaces, etc.
- *Data set statistics*. When managing huge collections, one could consider including some precomputed statistic about what follows in the data sets. 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.
- *Other information*. A provider can take into account other metadata for the understanding and management of the data.

### 2.3 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 [15]. 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 data set. 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 data sets are supposed to be managed by automatic processes, so that a more effective replacement can be done. The Dictionary component assigns a unique ID to each element in the data set. This way, the dictionary contributes to the goal of compactness, by replacing the long repeated strings in triples by their short IDs. In fact, the assignment of IDs, named as mapping [7], is usually the first step in RDF indexing.

The sets of subjects, predicates and objects in RDF are not disjoint. In order to assign shorter IDs, we distinguish between four sets (in an RDF graph $G$):

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

Figure 2 shows an example of these four sets within a dictionary building process. Note that a given ID can belong to different sets, but the disambiguation of the correct set is trivial when we know if the ID to search is a subject, a predicate or an object. A similar partitioning is taken in some RDF indexing approaches [2].

The subject-object ratio defined in Section 2.1, $\alpha_{s-o}$, characterizes the ratio of the subject-object set in the dictionary, composed of nodes with out-degree and in-degree greater than 0, $deg^{-}(a), deg^{+}(a) > 0$. We have noted that, in those data sets with a noticeable value of $\alpha_{s-o}$, common subject-object identification has and advantage over a disjoint assignment, thus reducing the dictionary size. 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, supposing an implicit order of IDs and some form of distinction between sets.

The Dictionary component allows multiple configurations. 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. Prefixes and shared strings (specially for URIs) could be identified and written once and then reference the unshared portions incrementally. These design decisions should be declared in the Header component.

### 2.4 Triples

By means of the Dictionary component, an original RDF triple can now be expressed by three IDs, replacing each element in triples with the reference in the dictionary (ID-based replacement in Figure 2). As we transform a stream of strings into a stream of IDs, we can take advantage of some interesting properties.

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:

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

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 performs a subject ordered grouping, that is, triples are reorganized in adjacency lists, 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.

In the notation above, all the data is represented by one stream, in which the list of objects associated with a subject ($s$) and a predicate ($p$) is represented just after the $p$. Instead, we decide to split this representation 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, *e.g.* the non-assigned zero ID. The second stream (*Objects*) groups the lists of objects for each pair $(s, p)$. These pairs are 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) implies a change of $(s, p)$ pair, moving forward in the first stream processing.

Figure 2 exemplifies the subject grouping and the final adjacency lists splitting into two coordinated streams. For instance, consider the list $[1, 2]$ in *Objects* stream. This

is the fourth list in the stream, so it refers to the fourth predicate in *Predicates*: the ID 3. This predicate belongs to the second list in the stream; that is, it is related with the second subject. Thus, the considered list develops the triples $(2, 3, 1)$ and $(2, 3, 2)$.

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

This proposal leads to a compact dictionary-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. In the next section we introduce appropriate structures to effectively implement the `HDT` proposal.

## 3 Compacting RDF with HDT: Practical Aspects

`HDT` allows RDF data sets to be represented in a compact form, with no restriction on how it should be implemented. This feature allows `HDT` to be optimized in specific applications. In this section, we approach a practical `HDT` implementation focused on RDF publication and exchange. The optimization is based on high `HDT` compressibility and efficient management processes.

### 3.1 Implementation

The final `HDT` comprises the concrete implementation of the three complementary representations of the *header*, the *dictionary*, and the set of *triples*.

*Header.* Header information can include multiple types of metadata, and the selected configuration can vary between different data sets and different providers. In order to reach a mutual understanding between providers and consumers, in final implementation we restrict the Header to be one RDF-valid format and we provide a specific *hdt* vocabulary (`http://hdt.dcc.uchile.cl/hdt#`) to describe the Header through four top-level statements (containers): (1) *hdt:publicationInformation* describes publication, source and provider information, (2) *hdt:statisticalInformation* includes statistics about the data, *e.g.* the parameters defined in Section 2.1, (3) *hdt:formatInformation* groups the specification of the location and concrete Dictionary and Triples representation, and (4) *hdt:additionalInformation* contains further information given by the provider.

*Dictionary.* The final Dictionary configuration is encoded on a single stream in which all strings (ended with a reserved character, *e.g* '`\2`') are concatenated. Thus, the sequence represents the order of the strings in their correspondent vocabulary of subject-objects (`S-O`), subjects (`S`), objects (`O`), and predicates (`P`). An empty word (also ended with the reserved character) is appended to the end of each vocabulary in order to delimit its size.

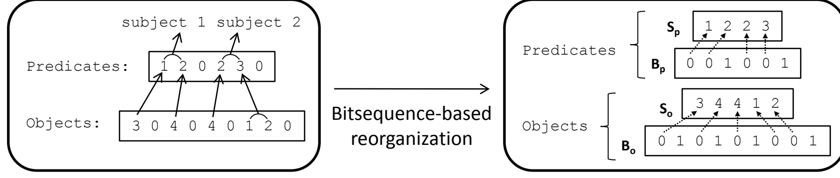



**Fig. 3.** Practical HDT Implementation

*Triples.* As we have previously explained, 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 embed in each stream to represent, implicitly, the graph structure. Our final implementation splits both parts in order to improve the HDT usability and to enhance its compactness. The graph structure is extracted from the original *Predicates* and *Objects* streams, so the 0-values can be deleted from them. The resultant sequences (respectively called $\mathcal{S}_p$ and $\mathcal{S}_o$) hold the original ordering for the positive integers. In turn, the graph structure is indexed with two bitsequences ($\mathcal{B}_p$ and $\mathcal{B}_o$, for predicates and objects) in which 0-bits mark IDs in the corresponding $\mathcal{S}_p$ or $\mathcal{S}_o$ sequence, whereas 1-bits are used to mark the end of an adjacency list.

Figure 3 shows a simple example of how the current approach reorganizes the original ID-based streams through the bitsequences. On the one hand, *Predicates*$=\{1,2,0,2,3,0\}$ evolves to the sequence $\mathcal{S}_p = \{1,2,2,3\}$ and the bitsequence $\mathcal{B}_p = \{001001\}$ whereas, on the other hand, *Objects*$=\{3,0,4,0,4,0,1,2,0\}$ is reorganized in $\mathcal{S}_o = \{3,4,4,1,2\}$ and $\mathcal{B}_o = \{010101001\}$.

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 length of the 0-bit sequences between two consecutive 1-bit represents the number of predicates in the corresponding list. For instance, the second 1-bit in $\mathcal{B}_p$ marks the end of the predicate adjacency list for the second subject ($P_2$). As we can see, a sequence of two 0-bit exists in between the previous and the current 1-bit. This means $P_2$ contains two predicates, which are represented by the third and fourth IDs in $\mathcal{S}_p$ by considering that the third and fourth 0-bit in $\mathcal{B}_p$ correspond to $P_2$. Thus, $P_2 = \{2,3\}$.

Data in $\mathcal{S}_o$ and $\mathcal{B}_o$ are related in the same way. Hence, the j-$th$ 1-bit in $\mathcal{B}_o$ marks the end of the object adjacency list for the j-$th$ predicate. This predicate is represented by the j-$th$ 0-bit 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 $(2,3,o) \in G$.

Each element in $\mathcal{S}_p$ and $\mathcal{S}_o$ is encoded, respectively, with a fixed-length code of $\log(|P_G|)$ and $\log(|O_G|)$ bits, by considering that the data set comprises $|P_G|$ and $|O_G|$ different predicates and objects. The bitsequences used to represent $\mathcal{B}_p$ and $\mathcal{B}_o$ make

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

1: $begin \leftarrow \mathbf{select_1}(B_p, s-1) + 2$;
2: $end \leftarrow \mathbf{select_1}(B_p, s) - 1$;
3: $size_{P_s} \leftarrow end - begin$;
4: $P_s \leftarrow \mathbf{retrieve}(S_p, 1 + \mathbf{rank_0}(B_p, begin), size_{P_s})$;
5:
6: $plist \leftarrow \mathbf{binary_search}(P_s, p)$;
7: $pseq \leftarrow \mathbf{rank_0}(B_p, begin) + plist$;
8:
9: $begin \leftarrow \mathbf{select_1}(B_o, pseq - 1) + 2$;
10: $end \leftarrow \mathbf{select_1}(B_o, pseq) - 1$;
11: $size_{O_{sp}} \leftarrow end - begin$;
12: $O_{sp} \leftarrow \mathbf{retrieve}(S_o, 1 + \mathbf{rank_0}(B_o, begin), size)$;
13:
14: $plist \leftarrow \mathbf{binary_search}(O_{sp}, o)$;

use of *succinct structures*. They are able to support `rank/select` operations over a sequence $\mathcal{S}$ of length $n$ drawn from an alphabet $\Sigma = \{0, 1\}$:

- $\texttt{rank}_a\texttt{(S,i)}$ counts the occurrences of a symbol $a \in \{0, 1\}$ in $\mathcal{S}[1, i]$.
- $\texttt{select}_a\texttt{(S,i)}$ finds the $i$-th occurrence of symbol $a \in \{0, 1\}$ in $\mathcal{S}$.

This problem has been solved using $n + o(n)$ bits of space while answering the queries in constant time [8]. We choose the *González, et al.* [12] approach to implement our bitsequences. This adds $5\%$ of extra space to the original length of $\mathcal{B}_p$ and $\mathcal{B}_o$, and achieves constant time for the required `select/rank` operations, which constitutes the basis for accessing to the structure of the graph.

### 3.2 `HDT` Management

A really huge RDF data set contains a volume of triples that becomes unmanageable when it is finally published. Let us suppose a very large data set has been published on any of the existing RDF syntaxes. Basic operations, *e.g.* check a triple existence or *access* to a subset of triples, are optimized in RDF storage systems, but this implies, firstly, configuring the system and then loading the full data set for the triple indexing. On the one hand, huge amounts of memory are required to render an efficient-time service when operating on the full data set. On the other hand, simple on-demand access to subsets of triples does not profit from the internal structure of RDF and suffers the cost of loading and searching the full data set.

Our current approach proposes an on-demand loading strategy able to take advantage of the structure indexed in $\mathcal{B}_p$ and $\mathcal{B}_o$ and accessible by fast `rank/ select` operations. A functional prototype is implemented in order to test basic operations. An initial stage loads both the dictionary (in a hash table) and the bitsequences; we consider that these structures always fit into memory. The sequences $\mathcal{S}_p$ and $\mathcal{S}_o$ remain stored in disk, queried by using the `Check&Find` operation described in Algorithm 1.

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 predicate IDs from $\mathcal{S}_p$. This operation seeks the position where $P_s$ begins in $\mathcal{S}_p$ (by using the $\texttt{rank}_0$ operation in line 4), and, next, retrieves the sequence of $size_{P_s}$ predicates that composed it. Once $P_s$ is available in memory, 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 rank in $\mathcal{S}_p$. In this step, we are able to retrieve the object adjacency list for $s, p$ ($O_{sp}$), by considering that 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 $s$: $\texttt{S}_p$) or in step `14` (the object $o$ is not in the object adjacency list for $s$ and $p$: $\texttt{O}_{sp}$). On the contrary, if $(s,p,o) \in G$, once the triple is found, the strings associated with $s$, $p$, and $o$ are retrieved from the dictionary in time $O(1)$.

In addition, the `Check&Find` operation sets the basis for building efficient insertion and deletion. In both cases, the adjacency lists to be updated are already available in memory after the `Check&Find`. Thus, the changes can be performed in an efficient logarithmic time, and the final performance of the operations will depend on the strategy for writing the updated information on disk. Besides, as we explain, `Check&Find` checks the triple existence, *i.e* the insertion is only performed if the triple does not exist and the deletion is carried out over the found triple.

We have assumed, in the initial step, that the dictionary fits into memory, a common assumption in the world of indexing regarding the size of the vocabulary. Our current development achieves reducing, in one order of magnitude, the original size by simply applying a prefix extraction. Other optimizations can be applied, such as a hierarchical treatment of URIs.

### 3.3 HDT Compression

RDF exchange is a common process in the global Web of data with the aim of sharing knowledge. The *data-centric* evolution of the Web will tend to demand even more exhaustive exchange processes in which efficiency is highly desirable. The performance of this task is directly related with the size of the data set, so large RDF data sets can overhead communication channels causing lengthy transmission delays. The use of universal compressors can alleviate this problem, although they are not able to detect and delete all the underlying redundancies of RDF.

We show, in Section 3.4, that our `HDT` representation (referred to as `Plain HDT` henceforth) is able to compact the RDF data set up to 15 times with respect to its

| Data set | Triples (millions) | Size (MB) | HDT | | Universal Compressors | | |
|---|---|---|---|---|---|---|---|
| | | | Plain | Compress | gzip | bzip2 | ppmdi |
| Billion Triples | 106.9 | 15081.74 | *31.87%* | **3.91%** | 9.54% | 6.83% | 5.32% |
| Uniprot RDF | 79.2 | 7083.22 | *14.33%* | **3.23%** | 8.71% | 5.04% | 3.99% |
| Wikipedia | 47 | 6882.20 | *6.62%* | **2.22%** | 6.97% | 5.11% | 4.10% |

**Table 1.** Compression results.

original size. This compacting ability proves capable of achieving very large savings in communication bandwidth and transmission delays. However, Plain HDT is even more compressible with very little effort. HDT-Compress makes specific decisions for each component:

**Header:** we keep this component in plain form as it should always be available to any receiving agent for processing.

**Dictionary:** it is compressed by considering that it stores all strings used in the RDF data set. Thus, we take advantage of repeated prefixes in URIs, specific $n$-gram distributions in literals, etc. This class of redundancy is able to be identified with a predictive high-order compressor. We chose PPM [9] to encode the dictionary.

**Triples:** the set of triples compression is independently attempted on each structure. On one hand, $\mathcal{S}_p$ comprises an integer sequence drawn from $[1, |P_G|]$. A Huffman [14] 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 5) 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[8] and ppmdi[9] to implement, respectively, the Huffman and PPM-based encoding.

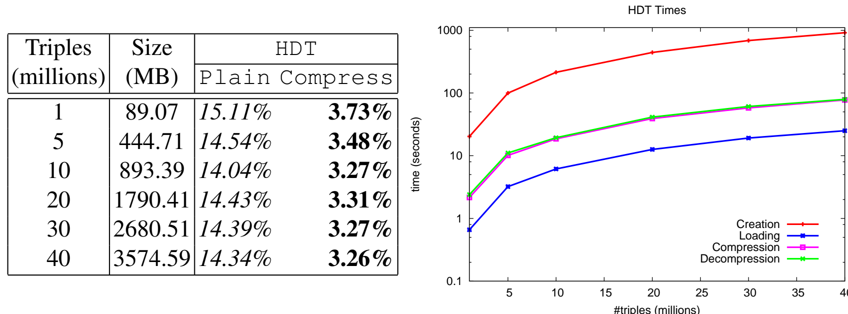
### 3.4 Experimental Results

This section shows the experimental results of the practical applications previously described for HDT. 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. We used a g++ 4.1.2 compiler with -O9 optimization. This experimentation was run on the data sets described in the final Appendix.

First, we study the HDT performance with incremental size of the Uniprot data set, from 1 to 40 million triples. This is shown in Figure 4. The left table studies the HDT effectiveness evolution. As can be seen, it is distributed between $14-15\%$ for Plain HDT, and by around $3.5\%$ for HDT-Compress (the percentage is always given with regard to the original file size). This proves the scalability of the HDT effectiveness by considering that the results do not directly depend on the size of the data set.

The right graph of Figure 4 shows relevant times for HDT. On the one hand, the *creation* time stands for the time required to transform an RDF data set (from plain

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



| Triples (millions) | Size (MB) | HDT | |
|---|---|---|---|
| | | Plain | 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%** |



**Fig. 4.** Performance of `HDT` (`Plain` and `Compress`) with incremental size data sets from Uniprot. The left table shows effectiveness, whereas the right figure draws significative times.

N3) 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`. This guarantees real time exchange processes by considering that the receiver is able to start the decompression as soon as the beginning of the compressed data set starts to arrive. In absolute terms, both compression and decompression times are slightly worse than the loading ones.

Table 1 compares `HDT` with respect to four well-known universal compressors. We choose `gzip` as a dictionary-based technique on LZ77, `bzip2` based on the Burrows-Wheeler Transform, and `ppmdi` as a predictive high-order compressor. We consider a heterogeneous corpora of RDF data sets shown in the final Appendix: Billion Triples, Uniprot RDF and Wikipedia with 106.9, 79.2, and 47 million of triples respectively.

The most effective universal compressors for all data sets 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 Wikipedia data set. 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 data sets. This supposes reductions between $3-4$ times with respect to `Plain HDT`, and consequently proportional improvements on exchanging processes. In turn, `HDT-Compress` also outperforms universal compressors by improving the best results, achieved on `ppmdi`, of between $20-45\%$.

## 4 Conclusions and Future Work

RDF publication and exchange at large scale are seriously compromised by the scalability drawbacks of current RDF formats and the lack of modular structure, internal metadata information and native operations over the data. `HDT` addresses these problems by approaching a more compact representation format, decomposing an RDF data source

into three main parts: Header, Dictionary, and Triples. Besides, this representation can be implemented by succinct structures and simple compression notions. This results in a very compact RDF representation able to support an on-demand `Check&Find` mechanism currently used to implement indexed access to the RDF graph. Our experimental results show the scalability of `HDT` for incremental data set sizes, being able to compact a data set up to 15 times current naive representations and providing efficient access to the data. In turn, a specific compression technique for RDF, `HDT-Compress`, outperforms universal compressors, which can serve as an essential choice in exchange processes involving huge data sets.

Current results open some interesting opportunities for future work. `HDT` compactness and the on-demand operations set the basis for developing an RDF storage system over `HDT`. The `Check&Find` mechanism and its ability to perform indexed access to the RDF graph will guide the design of efficient insertion and deletion (thus, also updating) which establish a full set of management operations. Additionally, we are currently analyzing $\mathcal{S}_p$ and $\mathcal{S}_o$ to be reorganized following a wavelet-tree-like strategy. This keeps the current spatial requirements of `HDT` but also provides indexed access inside both sequences, suggesting a full compressed index able to solve basic SPARQL queries natively. The resolution of a basic SPARQL join query can be performed through a series of wavelet-tree and bitsequences operations and dictionary accesses. Subject-object JOINs resolution can also profit from the common naming in the dictionary, as the elements are correctly and quickly localized in the top IDs.

## References


1. K. Alexander. RDF in JSON: A Specification for serialising RDF in JSON. In *SFSW 2008*, 2008. Available at http://www.semanticscripting.org/SFSW2008. Retrieved September 2010.
2. 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, 2010.
3. Dave Beckett. RDF/XML syntax specification (Revised). Technical report, W3C, February 2004.
4. P. Boldi and S. Vigna. The webgraph framework I: compression techniques. In *WWW 2004*, pages 595–602, 2004.
5. D. Brickley. RDF Vocabulary Description Language 1.0: RDF Schema. http://www.w3.org/TR/rdf-schema/, 2004. W3C Recomm. Retrieved September 2010.
6. F. Chierichetti, R. Kumar, and P. Raghavan. Compressed web indexes. In *WWW 2009*, pages 451–460, 2009.
7. E.I. Chong, S. Das, G. Eadon, and J. Srinivasan. An efficient sql-based rdf querying scheme. In *VLDB 2005*, pages 1216–1227, 2005.
8. D. Clark. *Compact PAT trees*. PhD thesis, University of Waterloo, 1996.
9. 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.
10. L. Ding and T. Finin. Characterizing the Semantic Web on the Web. In *ISWC 2006*, pages 242–257, 2006.
11. J.D. Fernández, C. Gutierrez, and M.A. Martínez-Prieto. RDF compression: basic approaches. In *WWW 2010*, pages 1091–1092, 2010.
12. R. González, S. Grabowski, V. Makinen, and G. Navarro. Practical implementation of rank and select queries. In *WEA 2005*, pages 27–38, 2005.

13. C. Gutierrez, C. Hurtado, and A. O. Mendelzon. Foundations of semantic web databases. In *PODS 2004*, pages 95–106, 2004.
14. D.A. Huffman. A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.
15. IBM. *IBM Dictionary of Computing*. McGraw-Hill, 1993.
16. D. McGuinness and F. van Harmelen. OWL Web Ontology Language Overview. http://www.w3.org/TR/owl-features/, 2004. W3C Recommendation. Retrieved September 2010.
17. T. Neumann and G. Weikum. RDF-3X: a RISC-style engine for RDF. *Proceedings of the VLDB Endowment*, 1(1):647–659, 2008.
18. T. Neumann and G. Weikum. Scalable join processing on very large rdf graphs. In *COMAD 2009*, pages 627–640, 2009.
19. 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.
20. J. Pérez, M. Arenas, and C. Gutierrez. Semantics and Complexity of SPARQL. *ACM Transactions on Database Systems*, 34(3):1–45, 2009.
21. 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.

## Appendix: The data sets used in the study

This appendix comprises an experimental study on real-world data sets in order to characterize RDF structure and redundancy by applying the parameters presented in Section 2.1. We chose three data sets based on the huge amount of triples, different application domains and previous uses in benchmarking:

– *Billion Triples Challenge:* One of the largest RDF data sets ($\sim$3.2 billion statements) available at the moment of writing, given as part of the Semantic Web Challenge[10]. Data is collected from Sindice, Swoogle, DBpedia and others.

– *Uniprot RDF*[11]: huge, freely-accessible RDF data set of protein sequence data, as part of the Uniprot project ($\sim$0.7 billion statements).

– *Wikipedia triple-set*[12]: English Wikipedia in RDF ($\sim$47 million statements).

A preprocessing step is first applied. Billion Triples data was parsed from N-Quads format[13] to NTriples by eliminating context information. We generated an N3 format from the original RDF/XML of Uniprot by using the tool SemWeb[14]. For Billion Triples and Uniprot, we used a random sample of the data, respecting the order of appearance and eliminating repeated triples.

Table 2 summarizes the statistical data, focusing on the most relevant parameters for our approach. Several comments are in order. First of all, note the high variability of values among the data sets. Billion Triple data set is a mashup of diverse sources, whereas Wikipedia triple-set and Uniprot are designed with one main purpose. The special condition of Billion Triple increments the number of different predicates, although

[10] http://challenge.semanticweb.org/
[11] http://dev.isb-sib.ch/projects/uniprot-rdf/
[12] http://labs.systemone.at/wikipedia3
[13] http://sw.deri.org/2008/07/n-quads/
[14] http://razor.occams.info/code/semweb/semweb-current/doc/index.html

| Data Set | # triples | # pred. | $deg^-$ | $\overline{deg^-}$ | $deg^{--}$ | $\overline{deg^{--}}$ | $degL^-$ | $\overline{degL^-}$ | $\alpha_{s-o}$ |
|---|---|---|---|---|---|---|---|---|---|
| Billion Triple | 106.9M | 50516 | 27387 | 2.74 | 27386 | 1.11 | 3293 | 2.46 | 6.54% |
| Uniprot RDF | 79M | 99 | 2030 | 4.80 | 2010 | 1.27 | 22 | 3.78 | 58.49% |
| Wikipedia | 47M | 9 | 7408 | 21.76 | 7400 | 3.95 | 7 | 5.51 | 17.61% |

**Table 2.** Data sets statistic summary.

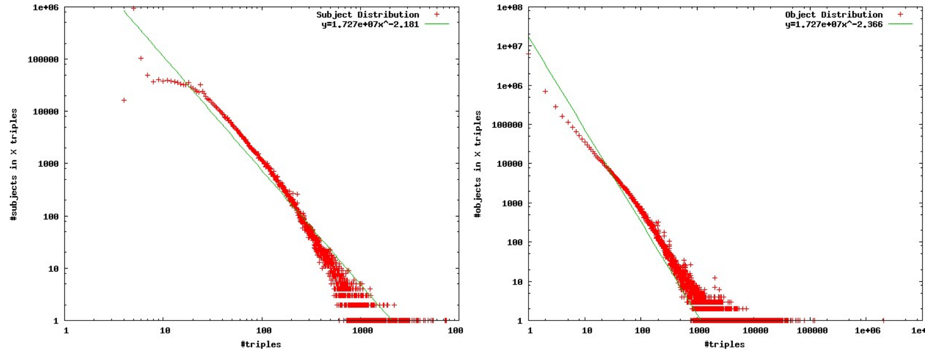



**Fig. 5.** Wikipedia triple-set distribution of subjects (left) and objects (right), *e.g.* a point (X,Y) in the rightmost graphic says that there are Y different objects each occurring in X triples. Both axis are logarithmic. The power laws have exponent $-2.181$ and $-2.366$ respectively.

they remain proportionally small to the number of triples, as well as decreasing the subject-object ratio. In this case, out-degrees reveal that subjects are related with few predicates (a mean of 2.46) and each of these pairs match with a single object (a mean of 1.11). In turn, the design of Uniprot has more cohesion, with a high subject-object ratio and a smaller number of very frequent predicates (each subject is related with a mean of 3.78 predicates over a total of 99). This reveals a star chained design in which a subject is strongly characterize and interlinked with others. A similar interpretation could be done for the Wikipedia triple-set, although the number of predicates is extremely low. In this case, the high partial degree suggests that a pair *(subject,predicate)* is repeated within several objects (a mean of 3.95). This affirmation is consistent with the interlinked design of pages in Wikipedia.

Figure 5 shows the distribution of subjects and objects of the Wikipedia triple-set. As we expected, a power-law distribution is remarkably present in both cases, suggesting an implicit significant redundancy. The other data sets reveal the same distribution for subject and object as well as a lack of statistical distribution of predicates.

These results immediately point to possible compact design models of RDF. Our approach, `HDT`, exploits the significant correlation and the inherent redundancy in data and structure. In particular, the Dictionary component takes advantage of subject-object ratio characterization and groups the references to the same node. The Triples component represents the graph compacting the distribution with implicit and coordinated adjacency lists, parametrized by out-degree means.