Binary RDF for Scalable Publishing, Exchanging and Consumption in the Web of Data

Javier D. Fernández

Supervisors:

Miguel A. Martínez-Prieto and Pablo de la Fuente
Department of Computer Science
University of Valladolid, Spain

Claudio Gutierrez
Department of Computer Science
University of Chile, Chile

DISSERTATION LEADING TO OBTAIN THE DEGREE OF DOCTOR IN COMPUTER SCIENCE

DataWeb Research. Department of Computer Science. University of Valladolid, Spain
Department of Computer Science, University of Chile, Chile

October 2013

Acknowledgments

First of all, I am aware that I am writing here one of the most read messages of this thesis. Thus, I am using part of these lines to do a little warning; you won't find the algorithm or the mathematical expression to solve all your problems, you won't even find conscientious reflexions opening minds. This thesis is just a story of how I, with the priceless support of my supervisors, spotted an open problem and carried out a research leading to efficient solutions. Much like the movies, the plot is a passionate travel throughout some unexplored fields and includes a good deal of sacrifice, hard working and, finally, success once the work obtained acceptation by the scientific community.

So these are the easiest acknowledgments I have ever written as I only have to list the main actors of this movie, of this research story. This thesis is almost exclusively due to Miguel A. Martínez-Prieto, Claudio Gutierrez and Mario Arias, who have been advisors, colleagues and friends. I could write two books as large as this one to describe our experiences together which are, now and forever, part of my life memories.

Thanks to Pablo de la Fuente for the support, and also to a full list of characters who don't even realize to be part of the plot in some way: Axel Polleres, Gonzalo Navarro, Rodrigo Cánovas, Diego Seco, Sandra Álvarez, Nieves Brisaboa and all LDB (Susana, Guillermo, Fari, ...), and a long long etcetera.

Thanks to those contributors to my life who do not expect a thank you; Victor, Diego, Sara, María, Pamela, Guillermo, Gonzalo, Gaby, Miguel, Irene, the Yuzzers and the old LAB216: Alejandro, Jorge, Zubi and Dani.

Thanks to the closest family in my heart, although some of them are already gone. Thanks to my parents who teach me everything, my brothers and sisters-in-law who stand me and love me despite my absences, my little nephew whom I hope to see grow being always ”his best friend“ and, of course, Estefanía, an undeserved gift, the reason of my happiness.

Finally, please remember again that this is just a little story, a minimum part of a never ending (and much better) saga of researching, but a story made with passion, the same passion Miguel and Claudio passed on to me and which I will try to spread in the next chapters.

Agradecimientos

Quisiera comenzar destacando que soy consciente de que escribo en estas líneas uno de los mensajes más importantes de esta tesis. Empleo por tanto este espacio para realizar una pequeña advertencia; posiblemente no encuentre el algoritmo o expresión matemática que solucione todos sus problemas, ni tan siquiera concienzudas reflexiones que expandan su mente. Esta tesis es tan sólo la historia de como yo, con el impagable apoyo de mis tutores, advertimos un problema abierto y llevamos a cabo una investigación para solucionarlo de la mejor manera posible. Al igual que en las películas, este guión es una viaje apasionado a través de campos inexplorados, e incluye una buena dosis de sacrificio, trabajo duro y, finalmente, éxito, ya que nuestro trabajo ha sido aceptado por la comunidad científica.

Así pues, estos son los agradecimientos más fáciles que nunca haya escrito, ya que únicamente tengo que listar los principales actores de esta película, de esta historia de investigación. Esta tesis se debe, casi en exclusiva, a Miguel A. Martínez-Prieto, Claudio Gutiérrez y Mario Arias, quienes han sido consejeros, colegas y amigos. Podría escribir dos libros tan largos como este mismo con nuestras vivencias juntos, que serán, ahora y por siempre, parte de mi memoria vital.

Gracias a Pablo de la Fuente por el apoyo, así como a una lista de personajes que ni tan siquiera son conscientes de su gran porte en esta película: Axel Polleres, Gonzalo Navarro, Rodrigo Cánovas, Diego Seco, Sandra Álvarez, Nieves Brisaboa y todo el LDB (Susana, Guillermo, Fari, ...), y un largo largo etcétera.

Gracias a aquellos contribuyentes a mi vida que no esperan las gracias; Víctor, Diego, Sara, María, Pamela, Guillermo, Gonzalo, Gaby, Miguel, Irene, los Yuzzers y el antiguo LAB217: Alejandro, Jorge, Zubi y Dani.

Gracias a mi familia que sigue cercana en el corazón, aunque algunos hayan partido ya lejos. Gracias a mis padres, quienes me enseñaron todo, a mis hermanos y cuñadas que me soportan y me quieren a pesar de mis ausencias, a mi sobrino a quien espero seguir viendo crecer como “su mejor amigo” y, por supuesto, a Estefanía, un regalo que no merezco, la razón de mi felicidad.

Finalmente, recuerde de nuevo por favor que esto es tan sólo una pequeña historia, una mínima parte de una saga de investigación sin fin, pero una historia hecha con pasión, la misma pasión que Miguel y Claudio me han transmitido y que trataré de transmitir a lo largo de los próximos capítulos.

Abstract

Current data deluge is flooding the Web with huge amounts of data represented in RDF, founding the so-called "Web of Data". Data about bioinformatics, geography, or social networks, among others, are already publicly available and interconnected in very active projects, such as Linked Open Data.

Several researching areas have emerged aside; RDF indexing and querying (typically through the SPARQL language), reasoning, publication schemes, ontology matching, RDF visualization, etc. Semantic Web topics related to RDF are, in fact, trending topics in almost every computing conference.

However, three facts can be gleaned from the current state of the art: i) little work is done in understanding the RDF essence before researching or applying this data model, ii) traditional RDF representations stay influenced by the old document-centric perspective of the Web, containing high levels of redundancy and verbose syntaxes to remain human readable. This leads to iii) fuzzy publications, inefficient management, complex processing and lack of scalability to further development the Web of Data.

In this thesis we first propose a deep study on the most important trends to face a global understanding of the real structure of RDF datasets. The main objective is to isolate common features in order to achieve an objective characterization of real-world RDF data. This can lead to better dataset designs, efficient RDF data structures, indexes and compressors.

Thereafter, we present our binary RDF representation, HDT, addressing the efficient representation of large RDF data through compact structures optimized for storage or transmission over a network. HDT partitions and efficiently represents three components of RDF data: Header, Dictionary and Triples. Next, we focus on dictionary and triple efficient structures, as long as they take part of HDT representation as well as most applications performing on huge RDF datasets. We propose novel techniques leading to compressed rich-functional RDF dictionaries and triple indexing. Finally, we propose the use of a succinct data configuration to browse HDT-encoded datasets. This structure holds the compactness of such representation and provides direct access to any piece of data.

Resumen

El actual diluvio de datos está inundando la Web con grandes volúmenes de datos representados en RDF, dando lugar a la denominada "Web de Datos". En la actualidad, se publican datos abiertos e interrelacionados sobre bioinformática, geografía o sobre redes sociales, entre otros, que forman parte de proyectos tan activos como *Linked Open Data*. Varias áreas de investigación han emergido de este diluvio; indexación y consulta de RDF (típicamente mediante el lenguaje SPARQL), razonamiento, esquemas de publicación, alineamiento de ontologías, visualización de RDF, etc. Los tópicos de la Web Semántica relacionados con RDF son, de hecho, *trending topics* en casi cualquier conferencia informática.

Sin embargo, podemos discernir tres importantes hechos del actual estado del arte: i) se han realizado aplicaciones e investigaciones apoyándose en datos RDF, pero aún no se ha realizado un trabajo que permita entender la esencia de este modelo de datos, ii) las representaciones clásicas de RDF continúan influenciadas por la visión tradicional de la Web basada en documentos, lo que resulta en sintaxis verbosas, redundantes y, aún, centradas en humanos. Ello conlleva iii) publicaciones pobres y difusas, procesamientos complejos e ineficientes y una falta de escalabilidad para poder desarrollar la Web de Datos en toda su extensión.

En esta tesis proponemos, en primer lugar, un estudio profundo de aquellos retos que nos permitan abordar un conocimiento global de la estructura real de los conjuntos de datos RDF. Dicho estudio puede avanzar en la consecución de mejores diseños de conjuntos de datos y mejores y más eficientes estructuras de datos, índices y compresores de RDF.

Posteriormente, presentamos nuestra representación binaria de RDF, `HDT`, que afronta la representación eficiente de grandes volúmenes de datos RDF a través de estructuras optimizadas para su almacenamiento y transmisión en red. `HDT` representa eficazmente un conjunto de datos RDF a través de su división en tres componentes: La cabecera (*Header*), el diccionario (*Dictionary*) y la estructura de sentencias RDF (*Triples*). A continuación, nos centramos en proveer estructuras eficientes tanto para el diccionario como para dicha estructura de sentencias, ya que forman parte de `HDT` pero también de la mayoría de aplicaciones sobre grandes volúmenes de datos RDF. Para ello, estudiamos y proponemos nuevas técnicas que permiten disponer de diccionarios e índices de sentencias RDF comprimidos, a la par que altamente funcionales. Por último, planteamos una configuración compacta para explorar y consultar conjuntos de datos codificados en `HDT`. Esta estructura mantiene la naturaleza compacta de la representación permitiendo el acceso directo a cualquier dato.

Disclaimer

I hereby declare that the work in composed by the candidate alone, except where explicitly indicated in the text.

This research herein was partially funded by the Regional Government of Castilla y León (Spain) and the European Social Fund, Erasmus Mundus, MICINN (TIN2009-14009-C02-02) and Fondecyt 1110287.

Contents

List of Figures

List of Tables

List of Algorithms

Why don't you find a place where there isn't any trouble? A place where there isn't any trouble. Do you suppose there is such a place Toto? There must be. It's not a place you can get to by a boat or a train. It's far, far away. Behind the moon, beyond the rain.

The Wizard of Oz (1939)

1 Introduction

1.1 Motivation

One of the main breakthroughs after the creation of the World Wide Web (WWW or simply the Web), was the consideration of the common citizen as the main stakeholder, *i.e.*, an involved part not only in the consumption, but also in the creation of content. To emphasize this fact, the notion of Web 2.0 was coined, and its implications such as blogging, tagging or social networking became one of the roots of our current sociability.

The complementary dimension to this successful idea deals with the machine-understandability of the Web. The WWW has enabled the creation of a global space comprising linked documents which express information in a human-readable way. The WWW has revolutionized the way we (humans) consume information. Agreeing this fact, it is also true, though, that its document-oriented model prevents machines and automatic agents from accessing to the raw data underlying to any web page. The main reason is that documents are the atoms in the WWW model instead of "data", hence data lack of an identity within documents.

A first approach to give structured meaning to data on the WWW was to incorporate machine-processable semantics to their information objects (pages, services, data sources, etc.). To fulfill these goals, the Semantic Web community and the World Wide Consortium (W3C) have developed models and languages for representing the semantics, as well as protocols and languages for querying it.

The Resource Description Framework (RDF) (Beckett, 2004) is the cornerstone of this semantic approach. RDF provides a graph-based data model to structure and link data that describes things in the world (Bizer, Heath, & Berners-Lee, 2009). Its semantic model is extremely simple; a description of an entity (also called resource) is represented through triples in the form *(subject, predicate, object)*. For instance, the two triples:

(wikipedia:Federico_García_Lorca, birthday, 5 June 1898)
(wikipedia:Federico_García_Lorca, friend_of, wikipedia:Pablo_neruda)

describe the entity *Federico García Lorca*, the famous Spanish poet, in Wikipedia. In the first triple, a value is given to its *birthday* property. The latter triple establishes a friendship relationship between the two famous poets.

An RDF dataset can be seen as a graph of knowledge in which entities and values are linked via labeled edges with meaning. These labels (the predicates in the triples) own the semantic of the relation, hence it is highly recommendable to use standard vocabularies or to formalize new ones as needed. This semantics are often defined using the RDF Schema (RDFS) (Brickley, 2004) and Web Ontology Language (OWL) (Hitzler, Krötzsch, Parsia, Patel-Schneider, & Rudolph, 2012). Typically, RDFS and OWL add a built-in vocabulary over RDF with a normative semantics.

Besides describing Web resources, the RDF Recommendation (Beckett, 2004) also devises a broader scope of application 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 increased adoption, has made RDF evolve from a simple model to represent metadata to a universal data exchange format.

In less than a decade, massive publication efforts have flooded the Web with very large RDF datasets from diverse fields such as bioinformatics, geography, bibliography, media and government data. This “democratization” in the creation of semantic data has being mainly driven by the Linked Open Data (LOD) community[1], which 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 (Bizer, Heath, Idehen, & Berners-Lee, 2008). It relies on the following four rules:

1. Use Unique Resource Identifiers (URIs) for naming resources;
2. Use HTTP URIs so that people can look up those names;
3. Provide useful information using standards, such as RDF and its corresponding query language, called SPARQL (Prud’hommeaux & Seaborne, 2008), when someone looks up a URI;
4. Include links to other URIs so that they can discover other related resources on the Web.

This philosophy pushes the traditional document-centric perspective of the Web to a data-centric view, emerging a huge interconnected cloud of data-to-data hyperlinks: the *Web of Data*.

The Web of Data (Bizer et al., 2009) converts raw data into first class citizens of the WWW. It materializes the Semantic Web foundations and enables raw data, from diverse fields, to be interconnected within this data-to-data cloud. It achieves an ubiquitous and seamless data integration to the lowest level of granularity over the WWW infrastructure. It is worth noting that this idea does not break with the WWW as we know. It only enhances the WWW with additional standards which enable data and documents to coexist in a common space. The Web of Data grows progressively according to the Linked Data principles. Latest statistics[2] pointed out that more than 31 billion triples were published, with more than 500 million links establishing cross-relations between datasets.

This powerful trend can be seen as a side effect of current data deluge in many other fields. It is easy to find real cases of massive data sources, such as scientific data (Hey, Tansley, & Tolle, 2009) (data from large-scale telescopes, particle colliders, etc.), digital libraries, geographic data, collections from mass-media and, of course, governmental data (educational, political, economic, criminal, census information, among many others). Besides, we are surrounded by multitude of sensors which continuously report information about temperature, pollution, energy consumption, the state of the traffic, etc. Any information anywhere and in anytime is recorded in big and constantly evolving heterogeneous datasets which take part in the data deluge. Definitely, it is the *Big Data* trending topic era.

Among all possible definitions, we refer to Big Data as “*the data that exceed the processing capacity of conventional database systems*”, that is, they are too big, they move too fast, and they do not fit, generally, the relational model strictures (Dumbill, 2012). Under these considerations, Big Data is popularly seen as the convergence of multiple “V’s”:

Volume is the most obvious dimension because of the large amount of data continuously gathered and stored in massive datasets exposed for different uses and purposes.

Velocity describes how data flow, at high rates, in an increasingly distributed scenario.

Variety refers to various degrees of structure (or lack thereof) within the source data (Halfon, 2012). This is mainly due to Big Data may come from multiple origins, hence data follow diverse structural models. The main challenge of Big Data variety is to achieve an effective mechanism to link diverse classes of data differing in the inner structure.

[1] `http://linkeddata.org`
[2] `http://www4.wiwiss.fu-berlin.de/lodcloud/state/` (September, 2011)

Whereas volume and velocity address physical concerns, variety refers to a logical question mainly related to the way in which data are modeled to enable efficient integration processes. It is worth noting that the more data are integrated, the more interesting knowledge may be generated, increasing the resulting dataset **V**alue (another Big Data characteristic). Thus, semantic technologies such as RDF and Linked Data perfectly fit the needs of Big Data (Styles, 2012); the use of such a graph-oriented representation (together with rich-semantic vocabularies) provides a flexible model for integrating data with different degrees of structure, but also enable these heterogeneous data to be linked in an uniform way for publication, exchange and consumption of this **Big Semantic Data** at universal scale.

It is worth noting that, although each piece of information could be particularly small (the so-called *Big Data's long tail* (Anderson & Andersson, 2007; Bloomberg, 2013)), the integration within a subpart of this Web of Data can be seen as huge interconnected data. RFID labels, Web processes (crawlers, search engines, recommender systems), smartphones and sensors are potential sources of RDF data. Automatic RDF streaming, for instance, would become a hot topic, specially within the development of smart cities (De, Elsaleh, Barnaghi, & Meissner, 2012). It is clear that Linked Data philosophy can be applied naturally to these *Internet of Things*, by simply assigning URIs to the real-world things producing RDF data about them via Web.

In practice, these potentially huge datasets are encoded by means of traditional verbose syntaxes which are still influenced by its conception under a *document-centric* perspective of the Web. RDF/XML (Beckett, 2004), for instance, is functional enough to add small descriptions (metadata) to documents or to mark web pages, but carries the heavy verbosity of XML to describe huge corpora. Later on, representations like N3 (Berners-Lee, 1998), Turtle (Beckett & Berners-Lee, 2011) and RDF/JSON (Alexander, 2008), have improved in several respects the original format, yet they are still dominated by a human-readable view.

It becomes clear that RDF must deal with the aforementioned three "V's" which are increasingly present in the Web of Data. To do so, considering RDF under a pure data-centric perspective is indispensable. We identify three general processes whose performance has to be significant improved:

- **Publication**. An analysis of current RDF datasets published in the Web of Data reveals several undesirable features (Fernández, Martínez-Prieto, & Gutiérrez, 2010). First, metadata about the collection is barely present or it is neither complete nor systematic. The lack of information is such that a "potential consumer" almost has to guess what the content of a dataset is about, disregarding its exploration in cases where the effort of consuming it seems not to worth the challenge. This is even more noticeable for mashups of different sources. Second, the published RDF dumps are actually bulks with no structure, no design, no final user in mind. They resemble unwanted creatures whose owners are keen to be rid of them (Fernández, Martínez-Prieto, & Gutiérrez, 2010).

- **Exchange**. Once a client decides that it is worth to get a dataset, it is exchanged under the same principles of the WWW. Despite their size, RDF datasets are exchanged within the plain aforementioned formats (*e.g.* XML, N3 or Turtle), which yields to high bandwidth costs and network delays. Universal compressors, such as gzip, are commonly used over these syntaxes in order to save space, yet it implies a subsequent decompression process at consumer.

- **Consumption**. Here we can distinguish two different types of consumptions. The first scenario arises following the natural flow of the previous publish-exchange process. After a final user has downloaded a dataset, it has to be postprocessed for diverse purposes (analysis, integration with other sources, local query, visualization, etc.) In general, plain RDF representations force to fully post-process the dataset in order to make it useful for consumption. Even the most basic data operation (such as searching for a triple or retrieving the description of a given resource) has to deal with the lack of any internal structure in the file, thus parsing the whole data. A second scenario of consumption regards the case in which the final user wants to make online queries (typically with SPARQL) over the RDF data served by a publisher. In this case, the response time depends on the efficiency of the underlying RDF indexes at the publisher which, again, have to deal with inefficient RDF representations.

In summary, current RDF representations diminishes the potential of RDF graphs due to the huge space they take in and the large time required for consumption. Moreover, similar problems arise when managing less RDF data but in mobile devices; together with scalability and memory constrains, these devices can face additional transmission costs (Le-Phuoc, Parreira, Reynolds, & Hauswrth, 2010).

The presented state of affairs does not scale to a *machine-understandable* Web of Data where i) large datasets are produced and published dynamically and ii) limited devices (mobile, sensors, Internet of Things) are increasingly joining this community.

1.2 Hypothesis

The motivation and current state of the art call for a binary representation for RDF aimed at reducing the high levels of verbosity/redundancy and weak machine-processable capabilities of the datasets. At the *physical level*, the binary RDF representation should permit efficient processing, management and exchange (between systems and memory-disk movements) at large scale. Thus, it has to minimize redundancy while guaranteeing modularity at the same time. At the *operational level*, desirable features include native support for simple checks for triple existence (lookups) and other simple query patterns.

Our hypothesis can be summarized as follows:

Given an RDF dataset, potentially huge, a lightweight binary RDF can encode the data leveraging the skewed structure of RDF graphs for the purpose of (i) large spatial savings, (ii) easy and modular data-centric publication and parsing and (iii) data retrieval.

With this hypothesis, we called for the need to move forward RDF syntaxes to a data-centric view. We propose a binary serialization format, HDT, that modularizes the data and uses the skewed structure of big RDF graphs (Ding & Finin, 2006; Oren et al., 2008; Theoharis, Tzitzikas, Kotzinos, & Christophides, 2008) to achieve large spatial savings. We present, in the following, the main requirements for an RDF serialization format:

- **It must be generated efficiently from another RDF input format and easy to convert to other representations.** For instance, a data publisher having the dataset in a semantic store must be able to dump it efficiently into an optimized exchange format. Similarly, if the serialization format enables data traversing to be performed efficiently, the conversion process to another (potentially binary) format can be completed more efficiently.

- **It must rely on a clear publication scheme.** The format must hold a standard scheme to include metadata about the data publication and its content, together with information to retrieve the dataset.

- **It must be space efficient.** The exchange format should be as small as possible, introducing compression for space savings. Reducing size will not only minimize the bandwidth costs of the server, but also the waiting time of consumers that are retrieving the dataset for any class of consumption.

- **It must be ready to post-process.** A typical case is performing a sequential triple-to-triple scanning for any post-processing task. This can seem trivial, but is clearly time consuming when large data are post-processed at the consumer.

- **It should be able to locate pieces of data within the whole dataset.** It is desirable to avoid a full scan over the dataset just to locate a particular piece of data. Thus, the serialization format must retain all possible clues enabling direct access to any piece of data in the dataset. A desirable format should be ready to solve most of the combinations of SPARQL triple patterns (possible combinations of constants or variables in subject, predicates and objects). For instance, a typical triple pattern provides a subject, leaving the predicate and object as variables (and therefore the expected result). In such case, we pretend to locate all the triples that talk about a specific subject[3]. In other words, this requirement contains a succinct intention; data must be encoded in such a way that "the data are the index".

1.3 Contribution

The **main contribution** of this thesis is a novel binary RDF format, called **HDT**: Header-Dictionary-Triples, addressing publication, exchange and consumption (index/query) of RDF at large scale. `HDT` represents the information of an RDF dataset in three optimized components:

- A *header*, including all type of metadata describing a big semantic dataset.
- A *dictionary*, organizing all the identifiers (IDs) in the RDF graph. It 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.

Specific contributions are as follows:

1. *Theoretical framework of RDF structure.* First, we tackle the problem of understanding the real structure of huge RDF graphs. To that end, we perform a deep study on these graphs revealing the underneath structure and composition of the graph. The main objective is to isolate common features to achieve an objective characterization of real-world RDF data. This can lead to better dataset designs, as well as efficient RDF data structures, indexes and compressors.

   With this objective in mind, we propose specific parameters to characterize RDF data. We specially focus on revealing the redundancy of each dataset, as well as their compact and compression possibilities. Finally, these metrics are evaluated on an evaluation framework comprising fourteen datasets which cover a wide range of modelings. Detailed results are summarized in Chapter 5.

2. *Binary RDF Specification.* Based on our analysis of the current scalability drawbacks managing Big Semantic Data, we design, analyze, develop and evaluate a binary RDF format, called `HDT`. `HDT` is aimed at reducing the studied high levels of verbosity and redundancy in real-world RDF, enhancing machine-processable capabilities of the datasets. Thus, `HDT` implements and gives response and sense to our hypothesis. We provide careful details of the design of the `HDT` components (Header, Dictionary and Triples), their operations and use. As `HDT` acts as a container and it is flexible enough to allow multiple configurations, we provide a practical deployment for publication and exchange, as well as an `RDF/HDT` syntax specification. This syntax took part of a W3C Member Submission (Fernández, Martínez-Prieto, Gutiérrez, & Polleres, 2011), validating the need of a well-defined binary format.

3. *Compressed Rich-Functional RDF dictionaries.* Based on the previous `HDT` dictionary, specific techniques for RDF dictionaries are proposed. We focus on highly-compressed RDF dictionaries

[3]Note that this query can be used to dereference an entity in accordance to the third Linked Data principle.

with very efficient performance at basic lookup functionality. We first adapt existing techniques for compressed string dictionaries. The proposed solution, a novel RDF dictionary called $\mathcal{D}_{comp}$, excels in size (it achieves the best compression ratios in our evaluation) and performance (over traditional dictionaries in the state of the art). Besides, its space/time can be finely tuned thanks to the organization of subdictionaries in $\mathcal{D}_{comp}$. In addition, advanced dictionary functionality for SPARQL filtering is proposed.

4. *Compact RDF triple indexes.* We address compact triple indexes on top of `HDT`-encoded datasets. We propose the use of succinct data structures and compression notions to approach practical implementations. All these indexes are developed on top of a novel triple structure for exchanging, referred to as Bitmap Triples (BT). The BT encoding sees the graph as a forest of trees and codifies its structure in two correlated bitsequences. Then, we propose lightweight indexes built efficiently at consumption time. The final configuration of triple indexes at consumer is called BTWO*. We describe the algorithms for triple pattern resolution using these indexes and, more important, the costs are clearly detailed with the metrics proposed. All configurations are studied and evaluated on real-world scenarios. Important conclusions are listed in Chapter 14.

5. *Practical deployment of binary RDF.* With the previous successfully achieved objectives for dictionaries and triples, we focus on efficiently integrating both components. That is, `HDT` is serialized with $\mathcal{D}_{comp}$ and BT components, and the additional indexes of BTWO*, as well as the required in-memory structures of $\mathcal{D}_{comp}$, are built efficiently at consumer. This proposal is deployed and evaluated against existing solutions in the field of RDF stores. Our experiments show how `HDT` excels at almost every stage of the publication-exchange-consumption workflow and remains very competitive in query performance.

1.4 Thesis Structure

First of all, Chapter 2 provides background on describing and querying semantic data and the Web of (Linked) Data processes of publishing, exchanging and consuming. We also introduce the concept of Big Semantic Data and provide basic concepts on succinct data structures and compression.

After that, the remainder of this thesis is organized in five parts, each one corresponding to a particular contribution, and a final summarizing part. Each part is composed of three chapters: an introduction and state of the art of the problem, our specific proposal and its empirical evaluation, and a final discussion. In particular, these parts include the following contents.

Part I tackles the characterization of the RDF structure for the purposes of efficient encoding. Chapter 3 collects the most important works leading to understand the RDF structure at large scale. Preliminary results, showing skewed RDF data distributions, set the basic foundations for more efficient RDF representations. Then, Chapter 4 proposes simple and feasible metrics characterizing RDF datasets. We establish an experimental framework illustrating these metrics for real-world RDF datasets. The chapter ends with a study revealing these metrics in different domains. Finally, Chapter 5 summarizes the contributions of this part and analyzes its implications in diverse related fields as well as the connection with the subsequent chapters.

Part II describes our proposal of a binary RDF representation, optimized for publication and exchange within the Web of Data. Chapter 6 motivates the problem, revises the state of the art, and describes our concrete goal. Chapter 7 presents our proposal `HDT`. First, we make a conceptual description of the `HDT` components *Header*, *Dictionary* and *Triples*. Next, we detail the basic encoding for a practical implementation focused on publication and exchange. Finally, we set up an experimental framework and provide results on compact ability and scalability. Chapter 8 discusses the applicability of the proposal and the provided results.

Part III presents new structures improving the basic functionality of RDF dictionaries on compressed space. Chapter 9 introduces and motivates the use of this type of RDF dictionaries to optimize RDF stores as well as providing novel functionality to binary formats such as `HDT`. The state of the art revises previous works on RDF dictionaries and string dictionaries. On this basis, we set up the goals of a novel dictionary. Chapter 10 focuses on our approach: $\mathcal{D}_{comp}$, a compressed and modular RDF dictionary. After a conceptualization, we present its modular configuration, data structures and algorithms for the lookup operations. We detail advanced filtering and push-up operations which can now be performed on the dictionary as a previous step of a triple scanning. Different $\mathcal{D}_{comp}$ configurations are tested on a experimental framework designed to characterize the compressibility and performance of the approach. The obtained results are discussed in Chapter 11.

Part IV focuses on triple indexes. Chapter 12 revises the state of the art in triple indexes, focused on RDF native structures and scalable approaches. Chapter 13 makes a first approach to construct a basic triple index for `HDT`-encoded datasets, proposing the use of compressed succinct data structures. This simple index allows some patterns to be efficiently resolved. Next, we introduce additional indexes to resolve complex graph patterns. Compressibility and query performance are studied on a testbed.

Part V exploits the presented dictionary and triple indexes to allow exchanged RDF to be directly consumed. We propose an integrated solution for querying `HDT`-encoded datasets and, thus, for efficient encoding and consumption of large RDF data. The resultant approach is called `HDT-FoQ`: `HDT` Focused on Querying and it is presented in Chapter 15. Next, we evaluate the Publication-Exchange-Consumption workflow on a real-world setup, analyzing the performance of each step as well as the overall process. Finally, we test the performance of `HDT-FoQ` for SPARQL querying.

To conclude, **Part VI** provides a critical discussion of the thesis. Chapter 16 summarizes the contributions and suggesting future direction of the research.

Publications and other results of this thesis are listed in **Appendix A**.

Don't be trapped by old concepts, Matthew, you're evolving into a new life form.

Invasion of the Body Snatchers (1956)

2 Basic Concepts

2.1 The Semantic Web

Much can be said about the World Wide Web (WWW) and its unparalleled success. It is simply one of the greatest invention ever, part of our everyday lives. The Web has tremendously changed or influenced fields such as education, libraries, music and video distribution, shopping and advertising markets, medicine and, of course, the way we communicate with friends, partners and other businesses.

In order to understand some of the limitations or shortcomings of the current Web, one has to go back to its original conception. As stated by his creator, Tim Berners-Lee, *"the goal of the Web was to be a shared information space through which people (and machines) could communicate"* (Berners-Lee, 1996). He thought a global information space, a virtual blackboard to write and read, to share and communicate both people and machines. But, *what is the shared content? What is written in this blackboard?* Documents, and links between documents accessible via the Internet. As we talk of a global space of information these documents (also called resources) have to be globally identified and hence (i) the Universal Resource Identifiers (URI) (Berners-Lee, Fielding, & Masinter, 2005) are the primary and key element of the original Web architecture. The second element, obviously, is the protocol for writing and reading in this global space, that is, (ii) the Hyper Text Transfer Protocol (HTTP). Last, how these interlinked documents are represented, which conforms (iii) the Hyper Text Markup Language (HTML).

These pillars of the Web have driven human communication to levels never seen before. Thanks to the adoption of new technologies (such as server and client-side scripting, Javascript, Ajax, etc.), the so-called Web 2.0 (Musser & Oreilly, 2007) brought the democratization in web publishing under novel forms of user-generated content. Note that part of this content is directly and consciously created by users, such as blogs, websites, podcasts, etc., while other part is series of user interaction records (metadata), such as ratings, comments, shares, likes, tags, navigation and query logs, etc., of which, eventually, a user loses control. Thus, the "shared information space" is also a space of *meta* information, though equally useful. After analysis, clustering and other data mining processes, *meta* information is one of the basis of advanced recommendation systems and efficient search engines (Baeza-Yates, Hurtado, & Mendoza, 2007; Borges & Levene, 2000).

Another remarkable side-effect of the latest Web development (among others, such as user-generated quality, data curation, trust or privacy) is the current blurring concept of document. Nowadays, the Web is so flexible, interactive and dynamic that a new resource can be instantly created or completely changed based on provided parameters or context information. Moreover, the content is not "ready" beforehand but tends to be extracted from relational databases, external APIs or other services which, typically, manage structured content, yet providing a final media representation (text, audio, video, etc.).

Very interesting questions raise from these two side-effects. *Is it positive or negative for a machine to have so much meta and dynamic (yet structured) information? Is it easy or complex for a machine to communicate in this shared but fuzzy space?* In fact, one could argue that, despite all the great success of the Web, the original purpose of "machine communication in the Web" has been marginally achieved. Tim Berners-Lee stated that, as a future direction, machines could take a stronger part in analyzing the

Web, and solving problems for us (Berners-Lee, 1996). Remaining true that machines are currently acting in the Web, they stay far away from the idealism. Consider solving the question: *is there any correspondence between the studies of the president of the developed countries and the destination of financial support for research projects?* It is obvious that the information could be in the Web, potentially distributed in different websites, in different formats, but we have limited automatic understanding of text semantics (even worse for other media). The challenge was already posed beforehand as a condition to the aforementioned future direction: "data on the web must be available in a machine-readable form with defined semantics". Without semantics a machine can hardly resolve such general and more complex questions and, in general, any task involving resource integration from different sources in the Web.

Fortunately, meta information and the underlying structured information are two sources a machine can better deal with. *How can we exploit the meta content and structured content for improving machine interaction with the Web?* And, if possible, *is it enough with the three pillars of the Web?* These are matters of the Semantic Web.

The *Semantic Web* was proposed by Berners-Lee, Hendler, and Lassila (2001) as a complement of the current Web in order to be more "machine-processable". It enhances the current WWW with machine-processable semantics imprinted in their information objects (pages, services, protocols, etc.). Its goals are summarized as follows:

1. *To give semantics to information on the WWW.* Although Tim Berners-Lee conceptualized a Web with random associations (unlike fixed database schemas), in the early stages of the Web there was still one line of thought modeling the Web as a database, designing formal models of Web queries (Mendelzon & Milo, 1998). The idea of using database techniques did not succeed and information retrieval techniques have dominated, and currently dominates the WWW information processing. One could argue that, at that time, the database approach was too futuristic once the amount of structured data on the Web did not yet reach a critical level that currently does (Gutiérrez, 2011). For this reason, the Semantic Web picks up on some ideas of database techniques which are structured via schemas that are, essentially, one kind of metadata. In the Web, metadata give the meaning (the semantics) to data and allows, or stimulate, advanced operations such as structured query, that is, querying data with logical meaning and precision.

2. *To make semantic data on the WWW machine-processable.* Assuming that semantic could be embedded in the Web, the aim is to encourage automatic machine processing: agents can perform tasks that users have to currently perform with arduous manual processes. Ideally, this objective could be also extended to the initial step of providing semantics. That is, in the current Web, the semantics of the data is mainly structured by humans who create domain-specific schemas. This manual process has known limitations (Quesada, 2008) at Web scale, hence it is crucial to automatize the process of "understanding" (giving meaning to) data on the WWW, which is equivalent to develop machine-processable semantics.

In summary, to fulfill these goals, the Semantic Web community, hand in hand with the World Wide Consortium (W3C)[1], has developed i) models and languages for representing the semantics, and ii) an infrastructure for it, *i.e.*, protocols, query languages and specifications for consuming these semantic data; accessing, consulting, publishing and exchanging (Gutiérrez, 2011).

In the following, we briefly describe the most known models and languages to represent (§2.1.1) and query (§2.1.2) semantic data. Next, we will call attention to the most feasible implementation (or variant) of the Semantic Web, the so-called Web of (Linked) Data (§2.2).

[1]`http://www.w3.org`

2.1.1 Describing Semantic Data

The *Resource Description Framework* (RDF) was an initiative of the World Wide Web Consortium (W3C), originally intended to provide an extension of the Platform for Internet Content Selection (PICS) content selection (superseded by the Protocol for Web Description Resources, POWDER[2]). PICS was envisioned as a filtering system for the content of the Web, in order parents to protect minors from "indecent" content. Technically, it was based on ratings and labels defined by content providers and other third-parties and a system for parental filtering. The cornerstone of the initiative was to provide PICS labels to be readable by machines (the filter software). After several discussions inside and outside the W3C, it became clear that this idea was valid for several additional applications and hence the W3C conformed the Resource Description Framework working group (Miller, 1998).

The original objective was to generalize the idea of machine-readable labels and to support metadata on the Web. That was the basis of the novel Resource Description Framework (RDF). The mechanism should provide *labels* to services, but also "permit string and structured values, and some other nifty features". In addition to services, RDF functionality was extended to add small descriptions (metadata) to documents, to protocols, to mark web pages or, obviously, to describe services.

The initial W3C Recommendation of RDF (Lassila & Swick, 1999) defines it as a "foundation for processing metadata" and establishes that its broad goal is "to define a mechanism for describing resources". This conception is clearly influenced by a *document-centric* perspective of the Web as it is stated through some examples of RDF application, such as the description of page collections that represent a single logical document or the intellectual property rights of web pages.

Nevertheless, the focus rapidly evolved to new frontiers. The current RDF Recommendation (Beckett, 2004) already devises an evolution of RDF *"to allow data to be processed outside the particular environment in which it was created, in a fashion that can work at Internet scale"*. That is, the focus is widen to "data", to information exchanged between applications without loss of meaning (Manola & Miller, 2004).

In the following we describe the RDF data model as well as two vocabularies given (or more precisely, extending) its semantics.

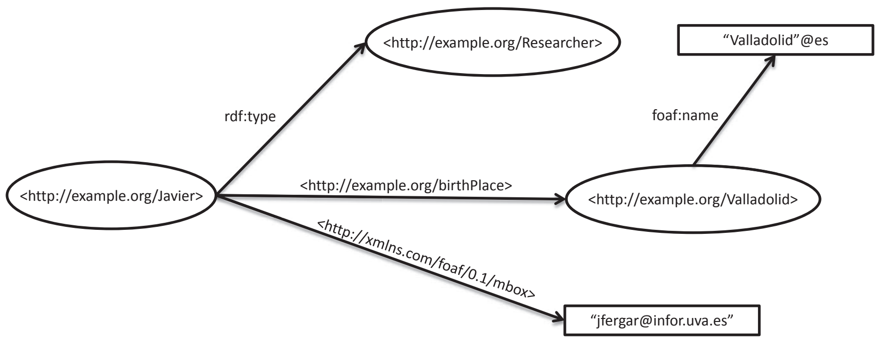
The RDF Data Model. It is implicitly built on two premises (Hogan, 2011):

- the *Open World Assumption* (OWA). In the open world we assume that any statement that is not known to be true is just "unknown" and not necessarily false (as would be assumed in the closed world systems such as relational databases). For instance, if we model the studies of the presidents of the developed countries and no studies are given for a particular president, let us say X, the closed world assumes X has no studies (one could imagine a *NULL* value in a relational database) whereas for the open world it is just unknown. Given that RDF aims to scale to the Web, it makes sense to assume that the information is potentially incomplete or unknown (*e.g.* the studies of the president X can be described in a third-party website).

- the *no Unique Name Assumption* (UNA). The UNA presence means that different names refer to different entities. The lack of UNA in RDF assumes that different names (in this case, URIs) can refer to the same entities (resources). The implication is that, on the one hand, naming resources becomes more flexible avoiding a centralized naming service. On the other hand, agents evaluating the similarity of two entities can not trust in their names and must evaluate other mechanisms.

RDF aims at describing resources, but at this point one could be bewildered by the concept of resource. We have spotted that the initial concept of documents, protocols, web pages and services was extended to general data, always under the OWA and the lack of UNA assumptions. Thereafter, RDF

[2]`http://www.w3.org/standards/techs/powder`



<http://example.org/Researcher>
"Valladolid"@es
rdf:type
foaf:name
<http://example.org/Javier>
<http://example.org/birthPlace>
<http://example.org/Valladolid>
<http://xmlns.com/foaf/0.1/mbox>
"jfergar@infor.uva.es"

Figure 2.1: A first RDF example.

generalizes the concept of a "Web resource", which means a thing that can be identified on the Web (Manola & Miller, 2004). The contact information of an individual, city facilities, every relation in a social network or product specifications are just few examples of resources which can be described. There is no limitation whenever we talk of something with an identity.

RDF describes resources through properties and the values for these properties. The values for the properties can be either other resources or constant values (called literals). That is, the basic atom in RDF are triples (also called statements) of the form:

(subject, predicate, value)

in which the subject is the resource being described, the predicate is a property applied to it, and the value (also called object) is the concrete value for this property. For instance,

(Javier, e-mail, jfergar@infor.uva.es)
(Javier, birth, Valladolid)

draw two RDF triples. This can be seen as a graph of knowledge in which entities and values are linked via labeled edges, *i.e.* the predicates are the labels. Part of the success of RDF is due to this graph conception and its expressive power: a dataset in RDF represents a network of statements through natural relationships between data, by means of labeled edges. This is also a matter for OWA as the labeled graph structure underlying to the RDF model allows new semantics to be easily added in advance. In other words, graph flexibility allows for handling semi-structured information (entities having different levels of detail).

Note that, in the previous triples, we have broken the aforementioned basis of RDF, machine-friendly processing, identity and naming of resources. That is, clearly *Javier* is not a Web identifier and machine processing of the properties can be misleading. For instance, *birth* is confusing as it can be understood as the birthday or the birthplace and hence the expected value changes alike (it can be a date, a string with the place, a link to the place, etc.). A similar appreciation can be done with *e-mail*; although in this case its meaning is more obvious, different RDF sources could spell different variations (*email*, *mailbox*, *contactmail*) or different languages (*correo*, *courriel*). These are just few examples showing that the RDF data model requires formalization in order to facilitate machine processes.

Figure 2.1 draws the RDF graph of an extension of this example, after formalization. As stated, resources are named using URIs, hence the resource *Javier* is named as *http://example.org/Javier*. Predicates, in some sense, hold the meaning of the descriptions and relationships of the resources, exemplified in the previous misleading. Therefore, predicates are named with URIs and they can be described as resources themselves. Sets of predicates are organized in vocabularies which people re-use for naming the same type of descriptions. For instance, *e-mail* is further described with *http://xmlns.com/foaf/0.1/mbox* and this URI is a well-known way of naming an e-mail property as it belongs to the Friend of a Fiend (FOAF) vocabulary[3]. In order to shorten URIs, prefixes are extensively used, such as *foaf:* which expands to *http://xmlns.com/foaf/0.1/*. Thereafter these are Compact URI (CURIE), although in the rest of the text we abuse of the language calling all URIs.

As shown in the figure, the objects in a triple can be another resources or literal attributes. Literals can be seen as end nodes[4], concrete values describing the resources. They can be (i) *plain* strings (such as *"jfergar@infor.uva.es"*), which can include language tags (such as *"Valladolid"@es*) or *typed* strings where XML Schema Datatypes can be used (*e.g.* "83"^^xsd:int). In any case, they may not be used as subjects or predicates in other RDF triples (Manola & Miller, 2004) and they should be treated as constants. If some structure on the values is needed, one could create a new resource with a URI grouping them, or make use of a special kind of node (in this RDF graph) called *blank nodes*. These unnamed resources usually connects various parts of the graph without the need of a URI. They usually serve as parent nodes to a grouping of data such as:

(ex:Javier, ex:contactInfo, _:javierAddress)
(_:javierAddress, ex:city, ex:Valladolid)
(_:javierAddress, ex:street, "Paseo de Belen 15")
(_:javierAddress, ex:postalCode, "47005")
(_:javierAddress, foaf:mbox, "jfergar@infor.uva.es")

in which *_:javierAddress* groups the contact information of *ex:Javier* (we use the "ex" prefix for *http:example.org)*.

An important consideration is that blank node identifiers are just a way of referencing them inside one RDF graph, *i.e.*, it can be seen as a local naming and the same identifier in two graphs does not imply to be the same blank node. The representation and use of blank nodes is entirely dependent on the concrete syntax used (Mallea, Arenas, Hogan, & Polleres, 2011).

At this point, it is worth noting that RDF is a data model and it does not restrict the multiple serialization formats emerged in the last years, which will be presented along the thesis. Thus, RDF is typically formalized as follows (Gutiérrez, Hurtado, Mendelzon, & Perez, 2011). Assume infinite, mutually disjoint sets U (RDF URI references), B (Blank nodes), and L (RDF literals).

Definition 1 (RDF triple) *A tuple $(s, p, o) \in (U \cup B) \times U \times (U \cup B \cup L)$ is called an RDF triple, in which s is the subject, p the predicate and o the object.*

Definition 2 (RDF graph) *An RDF graph G is a set of RDF triples. As stated, (s, p, o) can be represented as a direct edge-labeled graph $s \xrightarrow{p} o$.*

The normative semantics for RDF graphs (P. Hayes, 2004) follows the concept of interpretation, entailment and other classical treatment in logic (Gutiérrez et al., 2011). Its RDF vocabulary includes few pre-defined keywords such as *rdf:XMLLiteral*, *rdf:List*, *rdf:Statement* or *rdf:Bag*. One of the most important built-in predicate is *rdf:type*, as it allows for creating classes within the RDF graph. In this context, a "class" stands for a group of resources sharing common characteristics. For instance, in the previous example it was stated that *Javier* was a type of *foaf:Person*. Although this basic mechanisms does not allow "advanced operations" (such as modeling hierarchies), the compromise is that the more expressive power of its vocabulary semantics, the higher computational complexity is required for processing such data (Gutiérrez et al., 2011). Assuming that RDF was designed to be flexible and extensible, additional vocabularies can be used to add semantics to classes and properties. Throughout the next items we briefly describe the two most successful approaches, the RDF Schema and the Web ontology language.

[3]`xmlns.com/foaf/0.1/`

[4]Literals can not be the subject in triples, only URI resources can be described.

The RDF Schema (RDFS) (Brickley, 2004). It adds a built-in vocabulary to RDF with a normative semantics. That is, it provides a "basic type system for use in RDF models". These types are given within the same RDF data model and they deal with inheritance of classes and properties among other features. It can be thought of as a lightweight ontology.

Roughly speaking, the most noticeable contribution of RDFS vocabulary is to add four novel properties: *rdfs:subClassOf*, *rdfs:subPropertyOf*, *rdfs:domain*, and *rdfs:range*. Without going into details, the first two allows to define a basic hierarchy within classes and properties, whereas the latest delimit the class (or classes) of a subject or an object when they appear under a given predicate. For instance:

(ex:Researcher, rdfs:subClassOf, foaf:Person)
(ex:addressInfo, rdfs:subPropertyOf, ex:contactInfo)
(ex:addressInfo, rdfs:domain, ex:Researcher)
(ex:birthPlace, rdfs:range, ex:Place)

which models, first, that a *ex:Researcher* is a subtype of *foaf:Person* and the *ex:addressInfo* property is a type of *ex:contactInfo*. This states that a resource of type (*rdf:type*) *ex:Researcher* is also a *foaf:Person*. Similarly, a property value for *ex:addressInfo* is also attached to a *ex:contactInfo* predicate. The last two triples state that any resource related by a *ex:addressInfo* property is member of the class *ex:Researcher* (even though the type property is not explicitly given). In turn, a value given for the property *ex:birthPlace* is member of *ex:Place*. For instance, if we attach these triples to the example in Figure 2.1, a machine can automatically infer that *ex:Valladolid* is a *ex:Place*.

RDF semantics (P. Hayes, 2004) include entailment rules to make this type of deductions as well as the so-called RDFS axiomatic triples, *i.e.* axioms such as:

(rdf:type,rdfs:domain,rdfs:Resource)

Entailment rules can be seen as a deductive system (Gutiérrez et al., 2011) stating, for instance:

$$\frac{(A,sp,B)(B,sp,C)}{(A,sp,C)}$$

which describes the transitivity in subproperties (if the resource A is subproperty of B, and B is subproperty of C, then A is subproperty of C). Note that Gutiérrez et al. (2011) describe the complexity and bounds of the main problems.

The Web Ontology Language (OWL) (McGuinness & Van Harmelen, 2004). It is a version of logic languages adapted to cope with the Web requirements. Intuitively, it is more expressive than RDFS, allowing more advanced deductions yet, as stated, at the cost of computation complexity of evaluation and processing. Among all the novel language primitives, it highlights the following predicate:

(ex:Javier,owl:sameAs,dblp:Javier_D._Fernández)

because it allows for making equivalence between resources in different RDF graphs. In this case, the RDF graph example establishes a similarity with the external resource *dblp:Javier_D._Fernández* which (as we will explain) is part of the RDF graph of the bibliographic DBLP[5] catalog.

OWL comes in three flavors, at the cost of the aforementioned complexity: OWL Full, OWL DL and OWL Lite. In fact, the novel OWL 2 (Hitzler et al., 2012) adds new expressivity and redefines three new profiles, OWL 2 EL, OWL 2 QL, and OWL 2 RL.

In summary, describing semantic data remains a work in progress in which RDF is currently the cornerstone. Regardless of the novel potential fields of application, its most valuable attribute has always

[5]`http://www.dblp.org`

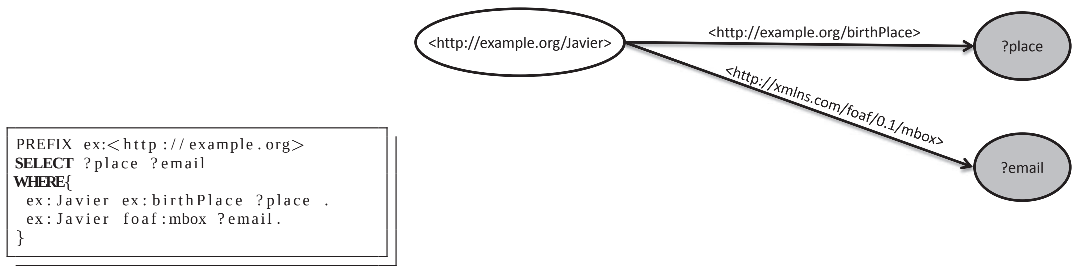
been its simplicity to serve as a mechanism for working with metadata which promotes the interchange of data between automated processes (Powers, 2003). Thus, if one has scalability in mind, due to complexity tradeoffs, the expressive power of the semantics should stay at a basic level of metadata.

In the following we present the most practical way (nowadays) of querying such semantic data.

2.1.2 Querying Semantic Data

RDF can be seen as a graph labeled with meaning, in which each triple (s, p, o) is represented as a direct edge-labeled graph $s \xrightarrow{p} o$. It is clear that a query language over the RDF data model should follow the same principles (interoperability, extensibility, decentralization, etc.) and a similar graph notion.

SPARQL (Prud'hommeaux & Seaborne, 2008) is the W3C recommendation for searching and extracting information from RDF graphs. It is essentially a declarative language based on graph-pattern matching with a SQL-like syntax, such as the one in Figure 2.2. This query retrieves the birthplace and e-mail of *ex:Javier* from an RDF graph such as the previous example (Figure 2.1). Intuitively, one should construct a graph pattern such as the one presented on the right, in which we provide named terms or variables if the term is unknown or part of the desired result. There exists solution when this graph pattern matches a subgraph of the RDF data after variable substitution. This required substitution of RDF terms[6] for the variables is then the solution for the query. The corresponding SPARQL query, with the appropriated syntax, is presented on the left side of the figure. The `WHERE` clause provides a serialization of the graph pattern to match against the data graph, whereas the `SELECT` clause lists which variables are given as results. In this case, the result is a simple "mapping", *?place="ex:Valladolid"* and *?email="jfergar@infor.uva.es"*, according to the original excerpt (Figure 2.1).



```
PREFIX ex:<http://example.org>
SELECT ?place ?email
WHERE{
 ex:Javier ex:birthPlace ?place .
 ex:Javier foaf:mbox ?email.
}
```

<http://example.org/Javier>
<http://example.org/birthPlace>
?place
<http://xmlns.com/foaf/0.1/mbox>
?email

Figure 2.2: A first SPARQL query.

In a general case, a SPARQL query $\mathcal{Q}$ comprises two parts, the head and the body. The head is an expression that indicates how to construct the answer for the query $\mathcal{Q}$ whose graph pattern is given in the body. In the previous query, the head makes use of a `SELECT` clause which select two variable as results. There are four output forms in total:

- `SELECT` which, as stated, allows for selections of matching values of the variables in the patterns.
- `ASK` are *yes/no* queries, *i.e.*, return *true* if the query pattern has a solution, or *no* in other case. Consider, for instance:

  $$\textit{ASK}\ \{\ \textit{?resource ?property ex:Valladolid .}\}$$

  which tests if there is something related to *ex:Valladolid* in the RDF graph. Over the graph in Figure 2.1, the result will be *yes*.

[6]An RDF term is a SPARQL terminology naming any element from $(U \cup B \cup L)$, though it extends URIs to IRIs (Duerst & Suignard, 2005).

- `CONSTRUCT` returns an RDF graph, as opposed to `SELECT` which returns a table of bindings for the variables. To do so, a graph template, which can include variables from the query pattern, must be provided. The substitution of these variables will provide the final RDF graph returned. In the following query,

  CONSTRUCT { *?resource ex:origins ex:Valladolid .*}
  WHERE { *?resource ex:birthPlace ex:Valladolid .*}

  we are constructing a simple graph in which the original *ex:birthPlace* predicate has been substituted by *ex:origins*. Note that the resulting graph only includes the triples described in the template, obviating the rest of the RDF graph. A query: *CONSTRUCT* { *?x ?y ?z .*} *WHERE* { *?x ?y ?z .*}, will return the original RDF graph.

- `DESCRIBE` returns an RDF graph with data about resources (Prud'hommeaux & Seaborne, 2008). It can be seen as a metadata request over the RDF graph. The concrete description is determined by the SPARQL query service holding the graph. One potential use is to know metadata information about a graph, *e.g.* the following query,

  DESCRIBE <http://example.org>

  returns a description of the graph which can include a summary of the type of resources included, authoring, relevant publishing dates, etc., which may be useful for an automatic process.

As stated, the SPARQL queries are built under the notion of graph pattern given in the body. The smaller component of a graph pattern is a triple pattern, *i.e.*, triples in which each of the subject, predicate and object may be a variable (this is formalized in Definition 3). The previous example showed two triple patterns, called a Basic Graph Pattern (BGP). In general terms, BGPs are sets of triple patterns in which all triple patterns must match (this is formalized in Definition 4). They can be seen as inner-joins in SQL. Several constructions can be applied over BGPs:

- BGPs can be grouped under braces.
- Alternatives of two groups can be expressed similarly to SQL, with a `UNION` keyword.
- Optional graph patterns can be provided with an `OPTIONAL` keyword.
- Matching values can be restricted by means of a `FILTER` clause.

The `OPTIONAL` constructor deals with the mandatory graph pattern matching. In BGPs, a solution is automatically rejected if just one triple pattern in a graph pattern (which can include several triple patterns) does not match. For instance, in the basic query in Figure 2.2, if *ex:Javier* does not include its birthplace in the original graph, the result will be completely empty. Instead, we would be interested in retrieving the e-mail in any case and, optionally the birthPlace if present. This is the goal of including optional parts, exemplified by the query in Figure 2.3. This query returns the emails of those individuals in the domain "infor.uva.es" and, if exist, it also retrieves their birthplaces. In other words, if the optional graph does not match, it returns no bindings but does not eliminate the solution (Prud'hommeaux & Seaborne, 2008), in tune with the principles of flexibility and *Open World Assumption*. It is worth noting that optional patterns have its relational counterpoint, the left outer join (Perez, Arenas, & Gutiérrez, 2009).

In turn, `FILTER` conditions are restrictions on solutions applied to a given group. They restrict solutions to those for which the filter expression evaluates to TRUE (Prud'hommeaux & Seaborne, 2008). They are *built-in* conditions often used to restrict the values of triples by means of several operators:

```
PREFIX ex:<http://example.org>
SELECT ?place ?email
FROM <http://example.org>
WHERE{
  ?someone foaf:mbox ?email.

  FILTER regex(?email,"@infor.uva.es")

  OPTIONAL {
        ?someone ex:birthPlace ?place .
   }
}
ORDER BY ?place
```

Figure 2.3: A slightly complex SPARQL query.

- Regular expressions (*regex*).
- Common arithmetic expressions.
- Other boolean operators, such as `BOUND(?variable)` which test if a valid mapping has been found for such variable or `isURI(?variable)` testing if the variable is a URI.

The SPARQL standard (Prud'hommeaux & Seaborne, 2008) details a complete list of operators. In Figure 2.3 we restrict with *regex* to those e-mails including "@infor.uva.es".

The previous query in Figure 2.3 also showed two novel clauses. The `FROM` clause allows to specify the graph (or graphs) to be queried. If two or more `FROM` clauses are provided, the graph to be queried is based on the RDF merge of the graphs. The `ORDER BY` clause is similar to its SQL counterpart, and is part of the *solution modifiers*. These are operators which, once the output of the pattern has been computed, allow to modify these values. A solution modifier is one of (Prud'hommeaux & Seaborne, 2008):

- `ORDER BY`, used to order the solutions.
- Projection, by means of selecting the desired variables in the `SELECT` clause. Note that in the query from Figure 2.3, not all variables are selected, as ?someone is just used to construct the graph pattern.
- `DISTINCT`, which allows to restrict to unique solutions.
- `REDUCED`, very similar to `DISTINCT` but it allows the SPARQL processor to partially eliminate the duplicates. In other words, the results are partially or totally removed.
- `LIMIT`, which restricts the number of solutions in an SQL-like manner.
- `OFFSET`, used as a pagination service of the solutions in combination with `ORDER BY` and `LIMIT`. It causes to start generating solutions after the specified `OFFSET` number of solutions.

The evaluation of a query $\mathcal{Q}$ against an RDF graph $\mathcal{G}$ is done in two steps: i) the body of $\mathcal{Q}$ is matched against $\mathcal{G}$ to obtain a set of bindings for the variables in the body, and then ii) using the information on the head, these bindings are processed applying classical relational operators (projection, distinct, etc.) to produce the answer $\mathcal{Q}$.

We provide in the following a brief excerpt of the most important SPARQL features in algebraic way, following Perez et al. (2009). Let us introduce two differences from the previous RDF conceptualization. First, we should include a novel set, V of variables, disjoint from the aforementioned U (RDF URI references), B (Blank nodes), and L (RDF literals). Next, URIs are extended to IRIs (Duerst & Suignard,

2005) in SPARQL, then we change to a set I of RDF IRI references. Thus, an **RDF triple** $(s, p, o) \in (I \cup B) \times I \times (I \cup B \cup L)$. Assuming the binary operators UNION, AND, FILTER and OPTIONAL (and the precedence AND over OPTIONAL).

Definition 3 (SPARQL triple pattern) *A tuple from $(I \cup L \cup V) \times (I \cup V) \times (I \cup L \cup V)$ is a triple pattern. In fact, this is the mentioned cornerstone concept of triple pattern. It is worth noting that blank nodes in graph patterns act as non-distinguished variables (Prud'hommeaux & Seaborne, 2008). As stated, the semantics of blank nodes prevents from using them as "persistent" identifiers, hence blank nodes in patterns does not reference specific blank nodes in the RDF graph.*

Definition 4 (SPARQL Basic Graph pattern (BGP)) *A SPARQL Basic Graph Pattern (BGP) is defined as a set of triple patterns. SPARQL FILTERs can restrict a BGP. If B_1 is a BGP and R is a SPARQL* built-in *condition, then $(B_1$ `FILTER` $R)$ is also a BGP.*

Definition 5 (SPARQL graph pattern) *A SPARQL graph pattern is defined recursively as:*

1. *A SPARQL triple pattern is a graph pattern.*
2. *If P_1 and P_2 are graph patterns, then $(P_1$ `AND` $P_2)$, $(P_1$ `OPTIONAL` $P_2)$ and $(P_1$ `UNION` $P_2)$ are graph patterns.*
3. *If P_1 is a graph pattern and R is a SPARQL* built-in *condition, then $(P_1$ `FILTER` $R)$ is also a graph pattern.*

Perez et al. (2009) complete this formalization with more semantics (mappings, evaluation, etc.) and provides a deep study on complexity query evaluation. Anglés and Gutiérrez (2008) reveal that the SPARQL algebra has the same expressive power as Relational Algebra, although their conversion is not trivial (Cyganiak, 2005).

A final remark deals with the SPARQL version. The SPARQL Working Group inside the W3C has produced a new SPARQL 1.1 Recommendation (March 2013) (Garlik, Seaborne, & Prud'hommeaux, 2013). Although it includes many interesting features (nesting of `SELECT` expressions, navigational capabilities thought property paths, an entailment regime for RDFS and OWL2 or aggregates), the novelties of this version go beyond the purpose of this thesis.

2.2 The Web of (Linked) Data

The "Web of Data" is a "twist" of the Semantic Web, a concrete proposal to dissipate the misgivings of an initial idealization. The idea behind the Web of Data is that we need to move forward machine-accessibility of the knowledge of the Web by means of publication, exchange and consumption of (raw) data in the Web. Gutiérrez (2011) provides a general (abstract) definition:

> *The Web of Data is the global collection of data produced by the systematic and decentralized exposure and publication of (raw) data using Web protocols.*

At this point, we have presented the Semantic Web and the way data can be modeled semantically with RDF, extended with additional semantics (RDFS, OWL) and queried with SPARQL. However, despite the expressive power and possibilities of this "infrastructure", one could think that we still remain in isolated RDF datasets, knowledge bases with axioms about a concrete subject. Thus, *how can we take advantage of the different sources publishing semantic data?* And even more important, *how can this be extended to a Web scale?*

First of all, the concepts was already grounded. RDF graph structure is flexible enough to represent interactions and relationships between data. These relationships can be at different levels; in an *internal level*, we establish relations between data inside a dataset. For instance, in the previous example from Figure 2.1, we link *ex:Javier* and *ex:Valladolid* in a meaningful way. Later on, we added a triple *(ex:Javier,owl:sameAs,dblp:Javier_D._Fernández)* in which we relate this internal resource with the information of an external source, DBLP. That is exactly the kind of relation at an *external level*. This feature allows to establish meaningful links between different data sources in such a way that, at Web scale, we could conform a semantic net of machine-processing descriptions. In fact, this is what led to the development of the Linked Data initiative.

Tim Berners-Lee envisioned a way to bring these ideas to the Web, in a practical way. He clearly stated the aim in a W3C design issue (Berners-Lee, 2006):

> *The Semantic Web isn't just about putting data on the web. It is about making links, so that a person or machine can explore the web of data. With linked data, when you have some of it, you can find other, related, data.*

The idea is to leverage the WWW infrastructure to produce, publish, exchange and consume (raw) data and not only documents (web pages). These processes reflect the current WWW philosophy in the sense that they are done by different stakeholders with different goals, in different forms and formats and, obviously, in a distributed manner.

To do so, Linked Data is a set of best practices formalized under the following four rules:

1. *Use URIs as names for things*. As stated, URIs allows real-world entities, its relationships as well as any raw data to be unequivocally identified at universal scale, *i.e.*, in the global space of the Web of Data.

2. *Use HTTP URIs so that people can look up those names*. This decision leverages HTTP to retrieve all data related to a given URI. In other words, those names can be dereferenced, they can be navigated using HTTP.

3. *When someone looks up a URI, provide useful information, using standards*. This rule standardizes processes in the Web of Data. One of the main challenges is the meaningful relationships of this universe of data (Hausenblas & Karnstedt, 2010), and this is where the aforementioned semantic data make sense. RDF and SPARQL, together with semantic technologies previously described, defines the standards mainly used in the Web of Data.

4. *Include links to other URIs, so that they can discover more things*. It encourages to establish external links between different datasets, breaking down the isolation and materializing data integration. A link is done by simply adding new RDF triples linking two entities from two different datasets. This inter-dataset linkage enables the automatic browsing throughout the net.

These simple four rules provide the basis for raw data to be published, exchanged and consumed by combining the RDF model and HTTP URI-based identification. The added value is that it allows different "things" in different datasets to be connected (*e.g.* scientific data, social networks information, media, government data, etc.), at the most basic level of granularity (an RDF triple) and to ask questions not possible before (thanks to the structuredness and expressiveness of SPARQL).

Linked Data is decentralized, strictly speaking it provides just a guide for publishing data with these best practices, hence they could be applied also in private (closed) environments. This "branch" of Linked Enterprise Data (Wood, 2010) leverages the infrastructure to improve several enterprise processes. In particular, the integration of data and applications can be lightened thanks to the underlying RDF model, and publishing policies help in exposing and sharing product and business information.

Nevertheless, the most visible and successful example of adoption and application of Linked Data principles is the Linked Open Data (LOD) movement[7]. The philosophy is to promote semantic data to be released with Linked Data principles and under open licenses. Its authors state that:

Linked Data is about using the WWW to connect related data that was not previously linked, or using the WWW to lower the barriers to linking data currently linked using other methods.

Tim Berners-Lee added (in 2010) a “five-stars” rating system to encourage people (specially, governments) implementing the Linked Data principles under an open license:

1. Make your stuff available on the web (whatever format) but with an open license, to be Open Data.
2. Make it available as structured data (*e.g.* Excel instead of image scan of a table).
3. Use non-proprietary formats (*e.g.* CSV instead of Excel).
4. Use open standards from W3C (RDF and SPARQL) to identify things, so that people can point at your stuff.
5. Link your data to other people’s data to provide context.

Essentially, LOD builds a cloud of semantic data-to-data hyperlinks[8]. This cloud has hugely grown since its origins in May 2007. The first report pointed that 12 datasets were part of this cloud, 45 were acknowledged in September 2008, 95 datasets in 2009, 203 in 2010, and 295 different datasets in the last estimation, which is already out of date (September 2011). As stated in the introduction, this last systematic study reported more than 31 billion triples and more than 500 million inter-dataset links. LODStats[9], a project constantly monitoring the LOD cloud, reports (in May 2013) 870 datasets (and other 1416 with problems) having more than 62 billion triples. Other statistics can be found in the Linked Open Vocabularies[10], the Linking Open Data Cloud from CKAN[11] and the OpenLink Software’s LOD Cloud Cache[12].

All kind of fields are present in LOD, such as geography, life sciences, media or publications. It is worth mentioning the noticeable presence of government data and the existence of many cross-domain datasets comprising data from some diverse fields. In fact, DBpedia[13] is considered the nucleus for the LOD cloud (Auer, Bizer, Kobilarov, Lehmann, & Ives, 2007). DBpedia is an RDF conversion of the structured data of Wikipedia, published under the Linked Data principles. It is an interesting example of a big semantic dataset. In the following, we will briefly bridge the Web of Data and the current hot topic of Big Data.

2.3 Big Semantic Data

Big Data is one of the current trending topics in Computer Science. As we stated in the introduction, we are living a *Data Deluge* era in which data comes from almost every field, at high volumes and high rates. Assuming the philosophy of the Web of Data (flexible, distributed, at Web scale, etc.), and the very productive fields in which semantic data is being produced (bioinformatics, geography, media), *can we*

[7] `http://www.linkeddata.org`

[8] From now on, we will indistinguishable talk of Web of Data, Web of Linked Data, or just Linked Data, but we always refer to Linked Open Data (LOD).

[9] `http://stats.lod2.eu/`

[10] `http://lov.okfn.org/dataset/lov/`

[11] `http://datahub.io/group/lodcloud`

[12] `http://goo.gl/sDUiO`

[13] `http://dbpedia.org`

talk of Big Semantic Data? And, *would this concept influence Big Data in general?* To answer the first question we have to review again the principles of Big Data. Then, we will move back to the first origins of *eScience* to exemplify the answer to the second question.

In short, among the several Big Data definitions, we use this term to refer to "*the data that exceed the processing capacity of conventional database systems*" (Dumbill, 2012). Big Data result then in the convergence of the following so-called "three V's":

V̲olume refers to the huge datasets continuously produced, stored and managed. *Scalability* is one of the main challenges related to Big Data. It is worth noting that storage decisions influence data retrieval which will often be the ultimate goal. Under this perspective, the semantic datasets fit in this dimension as they can be potentially huge. DBpedia and other LOD datasets are conformed of hundreds of millions of triples. A dataset integrating large RDF corpus could reach billions of triples and terabytes of data.

V̲elocity describes how data flow at high rates, in a distributed scenario. Moreover, the final user expects management and querying to be performed as fast as possible, specially in real-time systems. This again, perfectly fits the Web of Data, where dereference operations, complete downloads of RDF datasets and SPARQL queries (with potential large results) are performed. A significant interesting and very active area in LOD is, in fact, *streaming data processing* as sensors are able to produce and automatically exchange RDF data.

V̲ariety refers to the different degrees of structure (or absence) within the dataset (Halfon, 2012). Big Data has to deal with the different formats and data models coming from several distant fields and sources. Managing Big Data variety should rely on mechanisms for linking (and integrating) diverse classes of data. We have already argued that RDF datasets both own the similar variety concern and it is in fact a solution due to its flexibility and extensibility.

A four "V" is often added in order to refer to the **V̲**alue of the data, *i.e.* how fast data can be processed to obtain a significant value. The more interesting knowledge can be generated, the higher dataset value. It is obvious that semantic data in the Web of Data can generate an enormous value once it allows to stick together different "things" in different and potentially distributed datasets, thanks to the established meaningful links.

Thus, in this thesis, we introduce the concept of *Big Semantic Data*, recently presented in our work (Fernández, Arias, Martínez-Prieto, & Gutiérrez, 2013).

Definition 6 (Big Semantic Data) *The term "Big Semantic Data" refers to the semantic data whose volume, velocity and variety exceed the computational resources available for its efficient management in a given system.*

Note that we do not restrict solely to huge systems. The difference is noticeable. Although one could think in terabytes or petabytes talking about Big Semantic Data, few gigabytes may be enough to collapse an application running on a mobile device or a limited personal computer. As we consider that similar dimensions and problems could arise in such scenario, the definition pretends to cover all scalability issues.

Finally, we questioned if Big Semantic Data could influence Big Data in general. Back to the origins of the **Data Deluge**, Jim Gray devised its effects in the Science (Hey et al., 2009). He stated that scientists were no longer interacting directly with the phenomena as he envisioned that they should perform instead complex computational processes for analyzing the large data captured by instruments or recollected from simulations. Gray named this form of science the fourth paradigm: the *eScience*. But *eScience* was indeed not an easy task. It has to deal with the complexities of scientific data creation or capture, *sharing* these data with other scientists, and finally processing and analyzing such data. To do

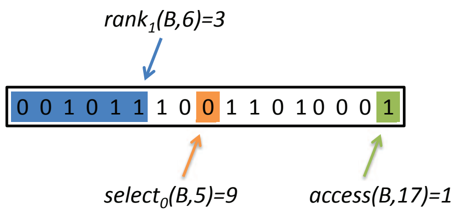


Figure 2.4: Example of a bitsequence $\mathcal{B}$ and `rank`/`select`/`access` operations.

so, Gray relied on machine-readable information. He stated that *"the only way that scientists are going to be able to understand that information is if their software can understand the information"*.

This example has its origins in Science but remains completely true in the current globalized Big Data scenario. It shows the importance of the *data representation* as one of the key factors in the process of creating, exchanging, storing, filtering, analyzing, and visualizing data at large scale. It is easy to find the correspondence between the words of Jim Gray and the use of the semantic standards (RDF, RDFS, OWL, SPARQL) previously presented. Big Semantic Data could actually influence Big Data whenever this datasets move to, partially shares or integrate, Web of Data models which allow advanced machine-processing facilities and leverage a complete Web-scale infrastructure for these data workflows. In addition, the graph-based model supports higher levels of variety before data become unwieldy, allowing more data to be linked and queried together (Styles, 2012).

2.4 Succinct Data Structures

The Big Data explosion has led to develop novel techniques, such as the well known MapReduce framework for data processing on distributed clusters (Dean & Ghemawat, 2008). In parallel, other "traditional" techniques have been reviewed to be adapted to the new reality. One of the main trends is to revisit data structures (*e.g.* trees, hashing or graph indexes) to take full advantage of the memory hierarchy. In other words, if data structures perform in higher levels of the memory hierarchy, the performance is clearly improved. While dealing with large data, one of the main requirements is that they need to represent and index as much data as possible taking minimum space and remaining performance efficient.

Recent years have witnessed a boom in compact structures with this latter purpose. These are the so-called *succinct data structures*, which are able to approach information theoretic minimum spaces while still serve efficient operations over the data. For instance, the compressed *full-text* indexes take space proportional to that used for the compressed text and replace it (Mäkinen & Navarro, 2007). A good example is the FM-index (Ferragina & Manzini, 2000) which counts the occurrences of an arbitrary pattern of length p in time $O(p\,log|\sum|)$, remaining close to the information theoretic minimum space.

The FM-index and most succinct data structures are based on `rank`/`select` operations over binary or arbitrary sequences (Mäkinen & Navarro, 2007). We briefly describe these operations over binary sequences and give references of the main practical implementations. Other advanced succinct data structures are described throughout the thesis.

2.4.1 Rank and Select over Binary Sequences

Given a sequence of bits $\mathcal{B}_{1,n}$, *i.e.*, a sequence of length n of bits, b, from an alphabet $\Sigma = \{0, 1\}$, three typical operations can be defined (a running example is shown in Figure 2.4):

- $\texttt{rank}_b(\mathcal{B}, i)$ counts the occurrences of bit b up to the *i*-th element, *i.e.*, in the prefix $\mathcal{B}[1, i]$. For instance, the operation in the example $\texttt{rank}_1(\mathcal{B}, 6)$ counts the number of 1-bits up to the sixth position (appearing in the prefix $\mathcal{B}[1, 6]$), resulting in 3.

- `select`$_b(\mathcal{B}, i)$ locates the position for the i-th occurrence of bit b in $\mathcal{B}$. Hence, `select`$_0(\mathcal{B}, 5)$ searches for the position where the 5-th occurrence of a 0-bit occurs which results in 9 in the example.
- `access`$(\mathcal{B}, i)$ returns the i-th element, *i.e.*, the symbol stored in $\mathcal{B}[i]$. The example shows the `access`$(\mathcal{B}, 17)$ operation which is a 1-bit.

Two additional operations are useful when iterations are made over bitsequences: `prev`$_b(\mathcal{B}, i)$ and `next`$_b(\mathcal{B}, i)$. These operations returns the position of the previous/next bit b from the i-th element, *i.e.* from $\mathcal{B}[1, i]$ or $\mathcal{B}[i, n]$ respectively. Nevertheless, these operations (as well as `access`) can be expressed via a constant number of `rank` and `select` queries (Mäkinen & Navarro, 2007).

In short, `rank` and `select` operations have been achieved attaching additional structures to the bitsequence with $o(n)$ extra bits of space while answering the queries in constant time (Clark, 1996; Munro, 1996). The idea (originally intended only for constant `rank` by Jacobson (1988)) is based on a two level directory of precomputed values and table lookups. In summary, given a bit array, a frequent operator is to count the number of set bits. This method uses precomputed tables storing these values for fixed length arrays. A fine tuning of the gap between counts at two levels (called superblocks and blocks) yields to constant time with the aforementioned $o(n)$ overhead.

González, Grabowski, Mäkinen, and Navarro (2005) provide two significant practical implementations. The first one follows the previous concept and uses a fixed 37.5% extra space on top of the original bitsequence size. The other practical implementation offers a space/time tradeoff. It uses just one level of precomputing and allows to parametrize the number of blocks. The more blocks, the more precomputed data and hence the faster performance at the cost of space. Each block takes $32 * k$ bits, *i.e.*, there are $\frac{n}{32*k}$ blocks and a total of $1/k$ space overhead. A common value is $k = 20$, still solving `rank` and `select` efficiently with just 5% space overhead. This implementation is referred to as `RG-k` in this thesis, where k is the mentioned parameter.

It is worth noting that none of these approaches takes into account the compressibility of the bits. In the words of Mäkinen and Navarro (2007), *although the $n + o(n)$ solutions are asymptotically optimal for incompressible binary sequences, one can obtain shorter representations for compressible ones*.

Among several practical representations, we highlight the approach from Raman, Raman, and Rao (2002). The underlying idea is that one could establish the most used bit configurations in blocks and to take advantage of the repetition when coding. A configuration can be represented as the number of 1-bits in the block and the concrete positioning of these bits inside the block. Then, every block is modeled with a tuple (c_i, o_i), where c_i is the so-called class (the number of bits) and o_i is the offset inside the list of possible variations for this number of bits. In this thesis we use the practical implementation of this method by Francisco Claude (*Compact Data Structures Library (libcds)*, 2012), referred to as `RRR-k` where k is the sample rate for partial sums. It performs in $O(k)$ time for `rank` and $O(\log len)$ for `select` where len is the length of the bitstring.

2.4.2 Rank and Select over General Sequences

The operations over binary sequences can be extrapolated to the context of a general sequence of symbols. In short, given a sequence $\mathcal{S}_{1,n}$ of n general symbols from an alphabet Σ of size σ:

- `rank`$_a(\mathcal{S}, i)$ counts the occurrences of $a \in \Sigma$ in $\mathcal{S}[1, i]$.
- `select`$_a(\mathcal{S}, i)$ locates the position for the i-th occurrence of the symbol $a \in \Sigma$ in $\mathcal{S}$.
- `access`$(\mathcal{S}, i)$ returns the symbol in $\mathcal{S}[i]$.

These operations over general sequences can be efficiently achieved by a structure called *Wavelet Tree* (Grossi, Gupta, & Vitter, 2003). A deep study on Wavelet Trees and their applications can be found in a recent work by Navarro (2013). In summary, a Wavelet Tree represents a general sequence of symbols as a balanced tree in which the alphabet, at each node, is split into "high" and "low" symbol values and the resulting subsequences are recursively subdivided until only one different symbol is present. Figure 2.5 shows an example of a Wavelet Tree over the sequence "one_mississippi". As can be seen, the

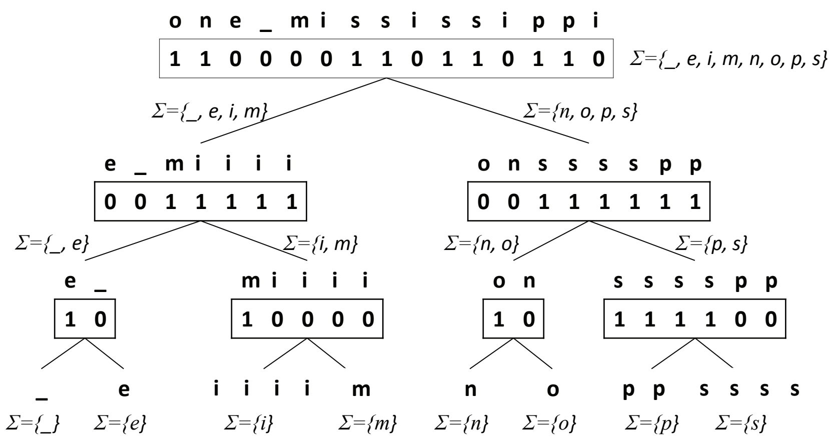


Figure 2.5: Example of a Wavelet Tree $\mathcal{S}$.

first level is split into two branches (or halves), one corresponding to lower symbols in $\{\_,e,i,m\}$ and the other for higher symbols in $\{n,o,p,s\}$. A bitsequence marks with a 0-bit the positions in the array belonging to the first halve, and sets a 1-bit when they belong to the second halve. The symbols in each level are shown only for illustration purposes, but only bitmaps are finally stored. Note that both the alphabet and the decision for splitting is known beforehand.

It is clear that this representation produces a tree of height $h = \lceil log\sigma \rceil$. Practical implementations answer `rank`, `select` and `access` in proportional time to its height h (Navarro, 2013). These operations over the Wavelet Tree are resolved making use of constant time `rank` and `select` operations of the underlying binary sequences (represented as bitmaps) in each level. We briefly detail these operation below over a running example shown in Figure 2.6:

- `access`$(\mathcal{S}, i)$ - Symbol at position i: To discern such symbol we have to navigate the tree from the given position on top to the symbol represented in leaves, in order. First, we start retrieving the bit b_i in the top level bitmap. If the retrieved value b_i is 0, we navigate to the left child branch, or to the right child otherwise. In the following level, we have to discount the number of previous positions that have gone to the other half. Thus, the position of the symbol we are looking for in the second level, $i_2 = rank_{b_i}(B, i)$. We continue descending in the three until the last level of leaves is reached. The symbol represented in the leaves is exactly the symbol at the position i.

  In the example in Figure 2.6, `access`$(\mathcal{S}, 9)$ asks for the symbol at position 9. As the bitmap at top level stores a 0-bit at such position, we descend to the left child. The novel position in the second level is $rank_0(B, 9) = 5$, thus we ask for the fifth symbol in the second level (as marked in the figure). The process continues and we finally descend to the leaves storing p-symbols, hence the symbol at the original position 9 was actually p.

- `select`$_a(\mathcal{S}, i)$ - position of the i-th occurrence of the symbol a: In this case we have to traverse the tree from bottom to top. First, we start in the i-th position in the leaves representing the symbol a. We climb up to the father node; the novel position in this node, let us say i_{h-1} (as it is in the $h-1$ level of the total h levels), is calculated as $i_{h-1} = select_{b_i}(B_{h-1}, i) = 7$, where b_i is 0 if the child node comes from the left half or 1 otherwise, and B_{h-1} is the bitmap at level $h-1$ for such path. This process is repeated until the top level is reached, in which the final operation $i_1 = select_{b_{i_2}}(B_2, i_2)$ returns the asked position.

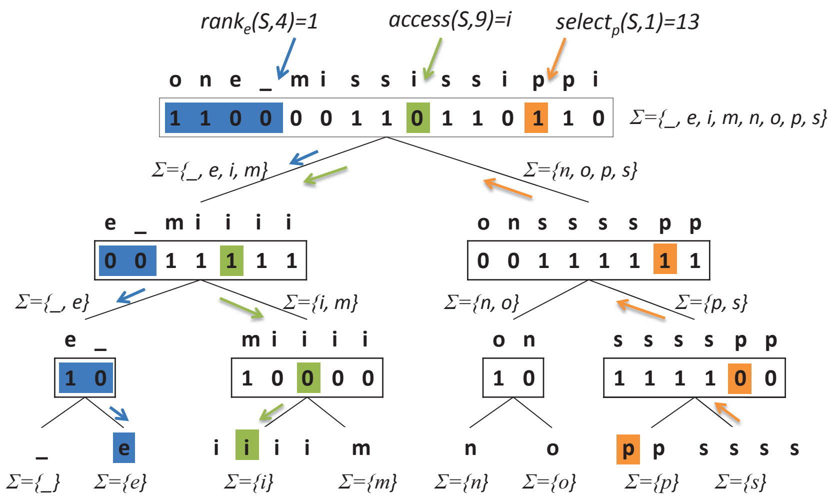


Figure 2.6: `Rank`/`select`/`access` operations over an example of a Wavelet Tree $\mathcal{S}$.

Figure 2.6 illustrates the resolution of $select_p(\mathcal{S}, 1)$, *i.e.*, we ask for the position of the first symbol "p". As explained, we proceed bottom-up, starting for the second position in p leaves. Note that this belongs to the first half (0-bits) of the split. Thus, the position in the father node (third level) is $i_3 = select_0(B_3, 1) = 5$. That is, we are positioned in the fifth position of the third level. As we are now in the second half of the vocabulary (1-bits), we climb up to the second level to a position $i_2 = select_1(B_2, 5) = 7$. Finally, we climb again throw the second half and hence the final position in the first top level is $i_1 = select_1(B_1, 7) = 13$.

- `rank`$_a(\mathcal{S}, i)$ - number of occurrences of a in $\mathcal{S}[1, i]$: We traverse the tree from top to bottom, delimiting at each level the range of positions we are interested in. We start descending to the second level by the appropriate branch b_1 given the symbol a. As we have to discount those symbols up to position i (we rename it i_1) that have gone to the opposite branch, the novel position in the second level is $i_2 = rank_{b_1}(B_1, i_1)$. We continue descending in the three until the last level of a leaves is reached. The position at leaves is exactly the number of a-symbols up to the original position i.

  Figure 2.6 shows the resolution of $rank_e(\mathcal{S}, 4)$, *i.e.*, we ask for the number of e-symbols up to the fourth position. Given that e belongs to the first half of the vocabulary, we descend to the second level by a 0-bit branch, and the maximum position at this second level is $i_2 = rank_0(B_1, 4) = 2$. Again, we descend to the third level by a 0-bit branch, to a position $i_3 = rank_0(B_2, 2) = 2$. Finally, the last descent is by a 1-bit branch and thus the final number of symbols is given by $rank_1(B_3, 2) = 1$. That is, only 1 e-symbol appears up to the fourth original position.

Practical implementations use $n\lceil log\sigma\rceil + o(n)log\sigma$ bits (Navarro, 2013). Note that $n\lceil log\sigma\rceil$ counts the total bits of the bitmaps (there are $\lceil log\sigma\rceil$ levels with at most n bits per level) whereas $o(n)log\sigma)$ holds the overhead to support intermediate `ranks` and `selects` in constant time.

As the previous extra space may be a problem on large alphabets (Navarro, 2013), a variant for the representation of levels has been proposed (Golynski, Grossi, Gupta, Raman, & Rao, 2007), hereinafter referred to as *GMR*. This representation draws a matrix $\mathcal{T}$ of $\sigma \times n$ bits, *i.e.*, one row per symbol and one column per position in the sequence $\mathcal{S}$. A 1-bit in the cell $\mathcal{T}[k, i]$ indicates that the symbol represented in the row k occurs in the position i of the sequence $\mathcal{S}$. Then, a bitmap A of size $\sigma \cdot n$ indexes this table by rows. Figure 2.7 illustrates the construction of a *GMR* structure over the string sequence of the previous example. One can easily see that A is highly compressible. Thus, A is logically split in blocks

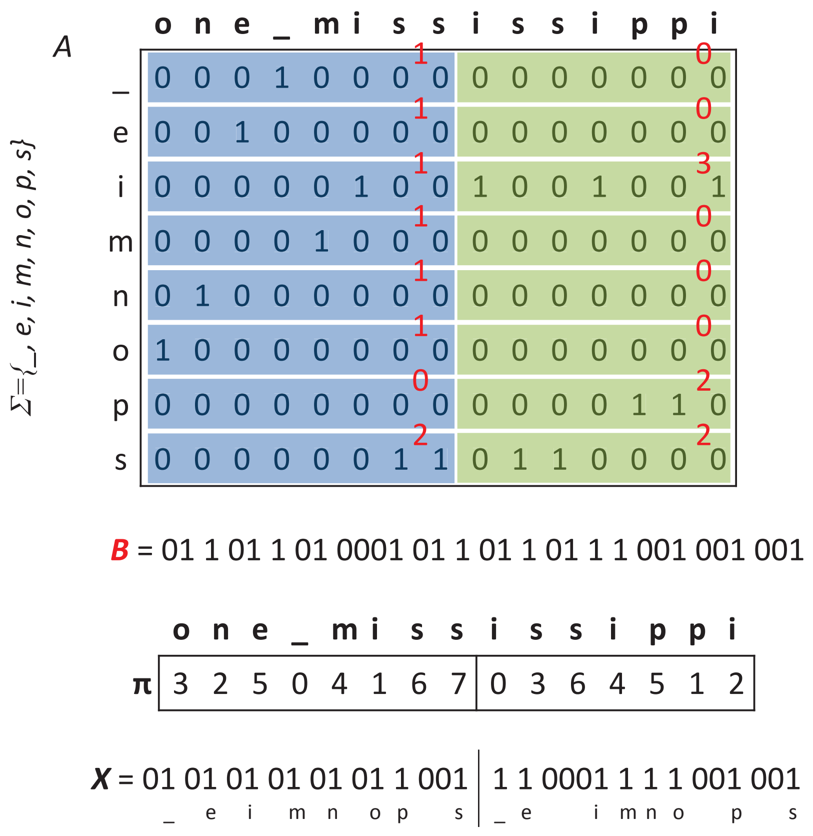

A
Σ={_, e, i, m, n, o, p, s}
o n e _ m i s s i s s i p p i
B = 01 1 01 1 01 0001 01 1 01 1 01 1 1 001 001 001
π 3 2 5 0 4 1 6 7 0 3 6 4 5 1 2
X = 01 01 01 01 01 01 1 001 | 1 1 0001 1 1 1 001 001

Figure 2.7: Example of a *GMR* construction.

of size σ: an additional array B stores the cardinality (number of 1-bits) in each block, in unary code. Note that the array B replaces A which is not stored. Two additional structures are then required: the first sequence lists, for each block, all the positions of each symbol in the block in alphabetical order. In fact, this is represented as a permutation, π, of the positions in the block $1, \cdots, \sigma$, as can be seen in Figure 2.7. The second sequence is a bitmap, $\mathcal{X}$, storing the cardinality of each symbol in the block, in unary. All this uses up to $nlog\sigma + o(nlog\sigma)$ bits and it provides the basis to perform `access`, `rank` and `select` with efficient performance. Without going into details, the resolution of these operations is based on first locating the corresponding block in $\mathcal{B}$ (restricted operations over each block are used), and then browsing the block with the structures π and $\mathcal{X}$. Finally, two variants are provided depending on the permutation encoding. On the one hand, one can support `access` and `rank` in $O(loglog\sigma)$, and `select` in $O(1)$. On the other hand, `access` can be revolved in $O(1)$ at the cost of $O(loglog\sigma)$ in `select`, and $O(loglog\sigma \cdot logloglog\sigma)$ in `rank`. More details can be found in Golynski et al. (2007).

2.4.3 Basic Compression Notions

Succinct data structures share the basis of traditional compression which aims at representing an original message in a reduced space. This section summarizes the classification of compression techniques and different measures of their efficiency.

At the most general level, one could classify compression techniques in *lossless* or *lossy methods*. In the first case, the decompressor returns an exact copy of the original message. In the latter, it may obtain the message with some differences. Text compression, for instance, requires to recover the exact original text, whereas video streaming may afford some losses. Both techniques are then classified according to the codification of the message and the type of data modeling (Martínez-Prieto, 2010; Salomon, 2007a).

Codification. Two compression families are traditionally defined (Bell, Cleary, & Witten, 1990):

- *Dictionary techniques* first build a dictionary of phrases which, in this context, are any sequence of consecutive symbols in the message. Next, compression is achieved by substituting the occurrence of phrases in the message by an index to the entry in the dictionary. Ziv-Lempel algorithms (Ziv & Lempel, 1977, 1978) are prototypical examples of dictionary-based compression.

- *Statistical techniques* are based on estimating the probability of a symbol, and to use shorter codes for the most frequent ones. Huffman codes (Huffman, 1952) and arithmetic codifications (Abramson, 1963; Pasco, 1976; Rissanen, 1976) are common representatives of these methods.

Note that estimating the probability of symbols leverages on obtaining a feasible model of the message. In statistical compression, a model defines what is a symbol in a message, and some specific properties (such as the number of occurrences), required for the subsequent codification. Thus, a statistical method is typically seen as a "modeling + encoding" process. In the following, we classify statistical compression according to different models.

Modeling. The modeling phase can be one of the following schemes.

- *Static models* take a fixed probability distribution, known by the compressor and decompressor beforehand. These probabilities do not depend on the current message, hence it can lose compression capabilities if the real probabilities strongly differs from the model. They are, though, a suitable option in several scenarios due to its simplicity and processing speed. General image compression (such as JPEG) is a widespread field of application.

- *Semi-static models*, in contrast, build a specific model for each message. Compressors perform a two-pass process. In the first pass over the whole message, statistics are extracted. Once the model with frequencies is built, it remains static in the second pass, where the message is encoded. Thus, a symbol is always encoded with the same code. The model is provided in a header which is first processed by the decompressor prior to the message. This is the model followed by the Huffman coding (Huffman, 1952).

- *Dynamic models*, also called *adaptive* models, also construct a specific model for each message, but they perform on a single pass. They start with an initial configuration and progressively update the model for each symbol read. In turn, the decompressor only receives the compressed text, as the model is totally dynamic. Note that symbol frequencies are varying while reading the message, and hence a symbol may be represented with different codes in the resultant compressed message. Thus, the decompressor has to replicate the model as decompression progresses, in the same way compression did. Dynamic models are flexible and adapt to the distribution at each state of processing, optimizing the bits used to codify each symbol. In contrast, the continuous updating adds an overload time. Ziv-Lempel algorithms (Ziv & Lempel, 1977, 1978), arithmetic encoding (Abramson, 1963) and other text compressors, such as PPM (Cleary & Witten, 1984), are good examples of these models.

In any case, the efficiency of compression techniques can be measured in terms of time and space. In the first case, the complexities of compression and decompression denote the behavior of a technique. Empirical performance, then, is measured as compression and decompression times (seconds, milliseconds, etc.). In turn, the space effectiveness evaluates the compressive capacity of a given technique. Let us present some traditional metrics (a complete description can be found in Salomon (2007a)). We assume an input message of n bytes and its compressed counterpart of c bytes. Consider also that the original alphabet can be represented with b bits per original symbol.

- **Compression ratio:** it represents the effectiveness computed as:

$$(\frac{c}{n}) \tag{2.1}$$

  A value of 0.7 means that the compressed data occupies 70% of the original size.

- **Bit per symbol (BPS):** it measures the mean number of bits used in compression to represent each original symbol, as follows:

$$(b \times \frac{c}{n}) \tag{2.2}$$

Part I

Characterizing the RDF Structure

3 Introduction

If the poem's score for perfection is plotted on the horizontal of a graph and its importance is plotted on the vertical, then calculating the total area of the poem yields the measure of its greatness. A sonnet by Byron might score high on the vertical but only average on the horizontal.

Dead Poets Society (1989)

This first part of the thesis studies the underlying RDF structure essence. We first motivate this study (§3.1), based in the fact that few works address real-world RDF characterization. Nevertheless, we review these and other related works (§3.2, §3.3 and §3.4), prior to our proposal in the next chapter.

3.1 Motivation

Throughout the thesis we focus on an efficient RDF representation addressing the most important scalability issues in the current Web of Data. To do so, one should study the real structure of RDF datasets, in order to take advantage of some of its features. That is a common methodology when modeling data structures aimed at solving real problems. However, despite RDF is being widely used, its structural properties are barely known and exploited in real-world deployments. This could be seen as a natural consequence of its adoption. First, plain RDF representations do not even pose the question as metadata was confined to small pieces of descriptions (see background in §2.1.1). Later, they evolved to add some grouping and features to "abbreviate" constructions, yet with the unique intuition of subject repetition (a review of current RDF serializations is presented in §6.2.1). Besides, many RDF stores serving SPARQL were developed on top of well-known relational schemas and indexes, such as B-trees. In such cases, one could argue that the necessary reflexion of the underlying model has been superficially addressed.

The objective of this chapter is to present the sparingly number of studies addressing real-world RDF structural characterization. In the next chapter, we will establish a minimum set of metrics for our purposes, and develop its empirical study. Note, though, that the study of the RDF structure has to deal with two important and correlated aspects:

- **Part-whole relationship**. This term distinguishes the study of the structure of a given RDF dataset or the consideration of the whole Web of Data as a network of networks (Y. Gil & Groth, 2011).
- **Schema-instance separation**. URIs in RDF provides a global naming scheme for resources. As stated, the semantics can be completed through languages such as RDFS (Brickley, 2004) and OWL (McGuinness & Van Harmelen, 2004). They provide schema-level information of classes, properties and relationships. These (lightweight) ontologies are used to be encoded together with RDF, hence the study of the structure and topology can consider the ontology structure independently of the instantiations (the pure RDF data).

For our purposes, we are actually interested in the structure of a unique RDF dataset, which can (or not) include the schema. This chapter also reviews, though, some specific works at schema level as well as some characterizations of the whole Web of Data in order to have a wider perspective of the problem.

3.2 Power Law Distributions. Scale-free Network

One of the first conclusions of initial RDF studies was the presence of power law distributions. 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. For example, letting a, c, β be constants:

$$f(x) = ax^{-\beta}, \text{ thus } f(cx) \propto f(x)$$

As can be seen, power law distributions are *scale-free*: multiplying by a constant, $f(x)$ remains proportional to $x^{-\beta}$.

Empirical observations of power law distributions in real networks, *e.g.* the WWW, have induced a new interest in fat-tailed degree distribution (Dorogovtsev & Mendes, 2003). Fat-tailed and scale-free structures are the Internet (Faloutsos, Faloutsos, & Faloutsos, 1999; Govindan & Tangmunarunkit, 2000), WWW (Albert, Jeong, & Barabasi, 1999), scientific citation nets (Redner, 1998) and nets of protein-protein interactions (Jeong, Mason, Barabasi, & Oltvai, 2001).

RDF graphs are actually not random graphs. In those, $P(k)$, the probability that a vertex has a degree k, does not follow a Poisson distribution. RDF graphs, instead, follow power law distributions in most of their metrics, as seen throughout the following observations.

Observations. Although power law distribution validation could be methodologically arguable[1], in practice it is assumed as a common characteristic of RDF real-world data. Ding and Finin (2006) crawled more than 300 million triples from 1.7 million documents[2], founding power law distribution in most of the considered metrics:

- The number of documents RDF documents per website.
- The number of triples per RDF document. They stated that most resources are described with two to ten triples. Whereas few triples are not very useful (a triple carries on little information), complex descriptions can be reduced by other means (*e.g.* pointing to a resource which groups other information).
- The use of instances of the defined classes and properties. In other words, more than the 97% of classes and 70% of properties are defined but never used.

Bachlechner and Strang (2007) collected more than 1.6 million Fiend-Of-a-Friend (FOAF) documents. Although they focus on demonstrating small-world phenomenon as shown in Section 3.3, they also addressed degree distribution. Note that the number of triples related to a subject is called *out-degree* and the number of triples related to a object is called *in-degree*. They study different communities inside FOAF, reaching similar conclusions; the cumulative in- and out-degree distributions for each community, as well as the entire network, follow power law distributions. For the entire network, the linear regression obtains an exponent of $\beta \approx -2.1$ for both in- and out- distributions. Average degree is 9.56 whereas the maximum is $7,739$, reflecting its skewed distribution.

In a more recent work, Ge, Chen, Hu, and Qu (2010) point out the absence of a macrostudy on the instance level in the Web of Data. In order to carry out this study, they first define the notion of *Object Link Graph*. This considers an undirected graph of related URI instances, either directly or through blank nodes paths (blank nodes are then removed[3] as well as literals). This graph holds also a power law distribution. This test was performed against 110.5M objects recollected from the Falcon search engine (Cheng, Ge, & Qu, 2008). The slope of the distribution was fitted to 2.84, a little larger than the ones of the traditional Web ($\beta_{in} \approx -2.1$ and $\beta_{out} \approx -2.7$ (Broder et al., 2000)).

The same work considers domain-specific structures for two well-known datasets such as DBpedia and Bio2RDF[4]. Similar conclusions are obtained, with power law presence (slopes between $2.52 - 2.59$).

[1]Some authors are reluctant to ratify a power law following the criticism of Clauset, Shalizi, and Newman (2009).

[2]Ding and Finin (2006) name Semantic Web documents (SWDs) to each pure RDF graph or Web page with embedded RDF graphs. We refer this simply as RDF documents.

[3]Note that blank nodes cannot be referred in other RDF graph as being the same node.

[4]`http://bio2rdf.org/`

Power law at the schema level. As stated, Ding and Finin (2006) also studied the schema level in their Web crawl, stating that 97% of classes and 70% of properties are never used.

Subsequent studies examined power law presence in more depth. Theoharis et al. (2008) studied the power law presence for 250 Semantic Web schemas, RDFS and OWL. They found power law distributions for about 58.6% of the schemas, for total-degree (sum of in- and out- degree) as well as for out-degrees (property domains) and in-degrees (property ranges), although the corresponding percentages are lower. Similar conclusions were inferred for the Discrete Random Variable (DRV) and the Cumulative Density Function (CDF) distributions.

Later, R. Gil and García (2004) confirmed the CFD fitting to power law. They performed an evaluation over 282 extracted ontologies (near 1.5M triples) from the DAML Ontology Library[5]. While Theoharis *et al* gave a range of $[0.65, 2.05]$ for the exponent of total degree distribution, Gil and García find a slope of 1.186, *i.e.*, centered on the previous range. Zhang (2008) obtained a slightly greater slope for two biomedical ontologies (FullGalen and NCI-Ontology), ranging in $2.12 - 2.47$.

One of the most recent works in this area Hu, Chen, Zhang, and Qu (2011) also confirm this distribution by recollecting $4,433$ ontologies in Falcons. They obtain a power law distribution for total degree with an exponent of 1.34.

Both for instance and schema level, (Guns, 2008) claims for quantifying the skewed degree distribution in more detail. This work proposes the use of the Lorenz curve (Lorenz, 1905), *i.e.*, the representation of the relative amounts $a_i = x_i / \sum x$ for i in $1..N$ being N the number of different elements in the distribution x. The y axis represents the cumulative fraction $a_1 + a_2 + \cdots + a_i$. The diagonal represents the case of perfect evenness (each case has the same amount). In-degree distribution is farther from the diagonal than the out distribution and thus it has more unevennesses.

3.3 Small-world Phenomenon

A graph is in fact a small world when it has short global separations, *i.e.*, the average minimum distance between nodes, L, is reduced (Watts, 1999). It is also associated with high local clustering (bigger than a random graph). The clustering coefficient, γ, for a vertex v, measures the probability that two neighbors of v are also neighbors in common. It is a measure of cliquishness of a network.

That is, formally defined, a small-wold graph having n vertexes with an average k degree and a characteristic path L when,

$$L \approx L_{random}, \text{ but } \gamma_{random} << \gamma, \text{ where } \gamma_{random} \approx k/n.$$

The small-world phenomenon has been popularly accepted within the networks of friends, stating that two random citizens are connected by only six degrees (intermediate nodes) of difference (Milgram, 1967). However, the consideration of the Semantic Web as a small world is still under discussion.

In practice, small-world networks have several important characteristics. Cliques (subgraphs in which all the possible connections are present) are highly represented, and most pairs of nodes will be connected by at least one short path (Bachlechner & Strang, 2007). This type of networks are also associated with a large presence of *hubs*, intermediate nodes with many associations, *i.e.*, high degree, and thus leading to power law distributions. These nodes are used to navigate through the network in fewer steps. They are good candidates for feeding them as seeds in the search engine (Ge et al., 2010).

In addition, as most nodes have small degree, small-world networks remain fault tolerant of random failures. However, major failures in the hubs may turn the graph isolated (unconnected). This is even more dangerous if the Web of Data is queried by automated agents with fewer recovering power. Addressing this issue, Guéret, Groth, Van Harmelen, and Schlobach (2010) propose metrics to evaluate robustness and to recommend optimizations, *i.e.*, nodes to add at the expense of fewer costs.

[5] `http://www.daml.org/ontologies/`

Once we have highlighted the most important works observing power law distributions[6], we review the most important studies on clustering and path length measures.

Clustering coefficient. Bachlechner and Strang (2007) questioned the small-world essence of the Semantic Web. As we stated, they collected more than 1.6M FOAF documents. They split the graph attending to FOAF communities, such as *TribeNet* or *LiveJournal*, evaluating the clustering coefficient in each community. They do find high coefficients in all subgraphs, for instance 0.168 versus $\gamma_{random} = 0.00024$ for *LiveJournal*, greater than the WWW factor of 0.108.

Later, R. Gil and García (2004) performed an evaluation at schema level. They studied 282 ontologies from the DAML Ontology Library. They computed the 1-neighborhood clustering coefficient for a directed graph and then they multiplied the mean value by two (in order to consider the graph as an undirected graph). The resulting clustering coefficient was 0.092, much greater than the corresponding $\gamma_{random} = 0.0000895$ for this case. The ontology clustering coefficient is slightly lesser than the WWW factor of 0.108 (Adamic, 1999).

Path lengths. Previous aforementioned works have also studied path lengths in the graph. Guns (2008), with a small corpus of instances, established the longest shortest path (diameter) in 11 whereas the average was only 4.12. The directed diameter of the Web is at least 28 (Broder et al., 2000) (for the connected component). Ge et al. (2010), with a bigger corpus, approximate an effective length to 11.53, which is still small regarding the size of the graph, but almost the double than the 6.83 for the traditional WWW[7](Broder et al., 2000). Bachlechner and Strang (2007) also found a value of 6.26 for its consideration of semantic network, near the 6.84 random value in theory.

Again R. Gil and García (2004), at the schema level, found 5.07 as the average path length, slightly lesser than the 6.83 for WWW. Cheng and Qu (2008) form a dependency graph of ontology terms and found also power law distributions and an average length path of 10.5. The recent work by Hu et al. (2011) studies the connectivity of the graph formed by matching ontologies. The clustering coefficient was 0.60 for classes and 0.72 for properties, whereas the average distance is 19.28 and 8.81 respectively.

3.4 Other Studies

The presented studies have shown that there exist several empirical studies working with different corpora at different time and different levels. All them, whether focused on the ontology or in a concrete instantiation, verify power law presence in graph degrees (essentially in- and out- degrees) and small-world criteria, $L \approx L_{random}$ and $\gamma_{random} << \gamma$.

Few studies leave this line of research and go into details. For instance, *what is the frequency of multivalued pairs (subject, predicate)? How many subjects act also as objects in other relations? Do typed subjects present different features?* None of these questions is addressed by previous studies.

Hogan et al.'s work (Hogan, 2011; Hogan, Harth, Passant, Decker, & Polleres, 2010) confirms many of those observations but additionally analyzes popularity in terms of interlinkage and publishing quality of RDF online, particularly focusing on compliance with Linked Data principles. Among statistical analysis, two relevant works, by Hogan, Polleres, Umbrich, and Zimmermann (2010) and the most recent by Hogan, Zimmermann, Umbrich, Polleres, and Decker (2012), define metrics such as cardinalities for (subject,predicate) and (predicate,object) pairs. These ran parallel to our research on RDF structure (Fernández, Gutiérrez, & Martínez-Prieto, 2010; Fernández, Martínez-Prieto, & Gutiérrez, 2010), hence some of these metrics are somehow considered in our proposal, fully detailed in the next chapter.

[6]power law presence is already an indicator of small-world graphs (Bachlechner & Strang, 2007).

[7]Considering the direction of links, average shortest-directed-path length between pages is equal to 16.

You still don't know what you're dealing with, do you? Perfect organism. Its structural perfection is matched only by its hostility.

Alien (1979)

4 Our proposal: Metrics for RDF Graphs

In this chapter we 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. As we motivate, our purpose is not to serve as a one-size-fits-all set of metrics, but to provide a simple set of useful metrics, a handbook toolkit when developing RDF data structures such as the ones we present in the next chapters. We also expect that some of these metrics and observations can provide insights to develop better dataset designs, other efficient RDF data structures, indexes and compression techniques.

4.1 Proposed Metrics

First of all, we note that RDF interpretation as a graph can be misleading. As shown in Definition 1 and 2, an RDF dataset can be represented as an edge-labeled graph. This conception is useful for some purposes such as modeling or visualization. However, it can not be considered a graph in the standard sense because the predicates can again appear as nodes of other edges (J. Hayes & Gutiérrez, 2004). Thus, the application of well-established methods from graph theory presents problems. For instance, traditional graph metrics must be reconsidered as well.

In the following, we provide specific parameters to characterize RDF data. We follow Perez et al. (2009) and Gutiérrez et al. (2011) for graph notation, with no distinction between URIs, Blank nodes and Literals[1].

4.1.1 Subject and Object Degrees

Previous studies (§3.2) focused on showing the presence of power-law distributions on subjects and objects. The presence of a skewed structure is a useful indicator as it points that some level of compression can be achieved (Salomon, 2007a). Thanks to the aforementioned concepts of succinct data structures, if compression can be achieved, there should exist a data structure with a good tradeoff between space and time performance. However, the design of these structures requires additional details, which is our purpose with the following metrics characterizing the concrete degree distribution in subject and object. Few indicators are sufficient, with simplicity in mind.

For the sake of clarity, we first summarize the purpose of each category prior to the formal definition:

- **out- and in- degrees:** to known the cardinality of subjects and objects. A subject with a high out-degree is a so-called *"star"* (a resource described in depth). An object with a high in-degree used to be a repeated final value or a hub to further information.

- **partial out- and in- degrees:** to describe the presence and cardinality of the multivalued pairs *(subject,predicate)* and *(object,predicate)*. That is to say, they quantify the number of objects related to the same *(subject,predicate)* and the number of subjects for a given *(object,predicate)*.

[1] 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 (Carroll, 2003).

- **labeled out- and in- degrees:** to know the number of different predicates related to subjects and objects. It is a mean showing if subjects are described with many or few predicates and, respectively, if objects are used with one or more predicates.

- **direct out- and in- degrees:** to count direct relationships between subjects and objects, thus minimizing the effect of the labeling. They consider to disregard labels and to count the number of objects related to a subject and, respectively, the corresponding number of subjects for each object.

Let G be an RDF graph, and S_G, P_G, O_G be the sets of subjects, predicates and objects in G. Assume generic $s \in S_G$, $p \in P_G$ and $o \in O_G$. Let us also denote Z_G and X_G the set of valid pairs *(subject,predicate)* and *(object,predicate)* respectively. That is, $Z_G = \{(s,p) \mid \exists z : (s,p,z) \in G\}$, and $X_G = \{(o,p) \mid \exists x : (x,p,o) \in G\}$.

Definition 7 (out-degree) *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.*

Definition 8 (partial out-degree) *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, $deg^{--}(s,p) = |\{(s,p,z) \mid (s,p,z) \in G\}|$. For the whole graph G, the* maximum partial out-degree, $deg^{--}(G) = max_{(s,p) \in Z_G}(deg^{--}(s,p))$, *and respectively the* mean partial out-degree *of graph G, $\overline{deg^{--}}(G) = \frac{1}{|Z_G|}\Sigma_{(s,p) \in Z_G} deg^{--}(s,p)$, are defined as the maximum (resp. the mean) partial out-degrees of all pairs of subject-predicates of G.*

Definition 9 (labeled out-degree) *The* labeled out-degree *of s, $deg_L^-(s)$, 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, $deg_L^-(s) = |\{p \mid \exists z \in O_G, (s,p,z) \in G\}|$. The* maximum labeled out-degree *of the whole graph, $deg_L^-(G) = max_{s \in S_G}(deg_L^-(s))$, and its corresponding* mean labeled out-degree *of the whole graph, $\overline{deg_L^-}(G) = \frac{1}{|S_G|}\Sigma_{s \in S_G} deg_L^-(s)$ of G, are defined as the maximum (resp. the mean) labeled out-degrees of all subjects of G.*

Definition 10 (direct out-degree) *The* direct out-degree *of s, denoted $deg_D^-(s)$, is defined as the number of different objects of G with which s is related as a subject in a triple of graph G. Formally, $deg_D^-(s) = |\{o \mid \exists y \in P_G, (s,y,o) \in G\}|$. For the whole graph G, the* maximum direct out-degree, $deg_D^-(G) = max_{s \in S_G}(deg_D^-(s))$, *and its corresponding* mean direct out-degree *value for graph G, $\overline{deg_D^-}(G) = \frac{1}{|S_G|}\Sigma_{s \in S_G} deg_D^-(s)$, are defined as the maximum (resp. the mean) direct out-degrees of all subjects of G.*

It is worth noting that, given the definition, the *direct out-degree* of a subject s can only differ from its *out-degree* when s is related to, at least, an object o by means of two or more different predicates. In other words, if every *(subject,object)* pair is only related with one predicate, then the *out-degrees* are equal to *direct out-degrees*.

Symmetrically, we define the *in-degrees* for objects in a formal way, as follows:

Definition 11 (in-degree) *The* in-degree *of o, denoted $deg^+(o)$, is defined as the number of triples in G in which o occurs as object. Formally, $deg^+(o) = |\{(x,y,o) \mid (x,y,o) \in G\}|$. The* maximum in-degree, $deg^+(G) = max_{o \in O_G}(deg^+(o))$, *and the* mean in-degree, $\overline{deg^+}(G) = \frac{1}{|O_G|}\Sigma_{o \in O_G} deg^+(o)$, *are defined as the maximum and mean in-degrees of all objects in O_G.*

Definition 12 (partial in-degree) *The* partial in-degree *of o with respect to p, denoted $deg^{++}(o,p)$, is defined as the number of triples of G in which o occurs as object and p as a predicate. Formally, $deg^{++}(o,p) = |\{(x,p,o) \mid (x,p,o) \in G\}|$. For the whole graph G, the* maximum partial in-degree, *$\underline{deg^{++}}(G) = max_{(o,p)\in X_G}(deg^{++}(o,p))$, and respectively the* mean partial in-degree *of graph G, $\overline{deg^{++}}(G) = \frac{1}{|X_G|}\Sigma_{(o,p)\in X_G} deg^{++}(o,p)$, are defined as the maximum (resp. the mean) partial in-degrees of all pairs of object-predicates of G.*

Definition 13 (labeled in-degree) *The* labeled in-degree *of o, denoted $deg^+_L(o)$, is defined as the number of different predicates (labels) of G with which o is related as object in a triple of G.Formally, $deg^+_L(o) = |\{p \mid \exists x \in S_G, (x,p,o) \in G\}|$. The* maximum labeled in-degree *of the whole graph, $\underline{deg^+_L}(G) = max_{o\in O_G}(deg^+_L(o))$, and its corresponding* mean labeled out-degree *value for graph G, $\overline{deg^+_L}(G) = \frac{1}{|O_G|}\Sigma_{o\in O_G} deg^+_L(o)$, are defined as the maximum (resp. the mean) labeled in-degrees of all objects of G.*

Definition 14 (direct in-degree) *The* direct in-degree *of o, denoted $deg^+_D(o)$, is defined as the number of different subjects of G with which o is related as an object in a triple of graph G. Formally, $deg^+_D(o) = |\{s \mid \exists y \in P_G, (s,y,o) \in G\}|$. For the whole graph G, the* maximum direct in-degree, *$\underline{deg^+_D}(G) = max_{o\in O_G}(deg^+_D(o))$, and its corresponding* mean direct in-degree *value for graph G, $\overline{deg^+_D}(G) = \frac{1}{|O_G|}\Sigma_{o\in O_G} deg^+_D(o)$, are defined as the maximum (resp. the mean) direct in-degrees of all objects of G.*

As previously stated, it remains true that if every *(subject,object)* pair is related only with one predicate, then the *in-degrees* are equal to *direct in-degrees*.

Note that *cardinality, average cardinality, inverse cardinality* and *average inverse cardinality* by Hogan, Polleres, et al. (2010) and Hogan et al. (2012) are equivalent to partial out-degree, average partial out-degree, partial in-degree and average partial in-degree.

Example and potential uses. Figure 4.1 illustrates these properties in a small example graph which is inspired by the previous example in Chapter 2 (Figure 2.1).

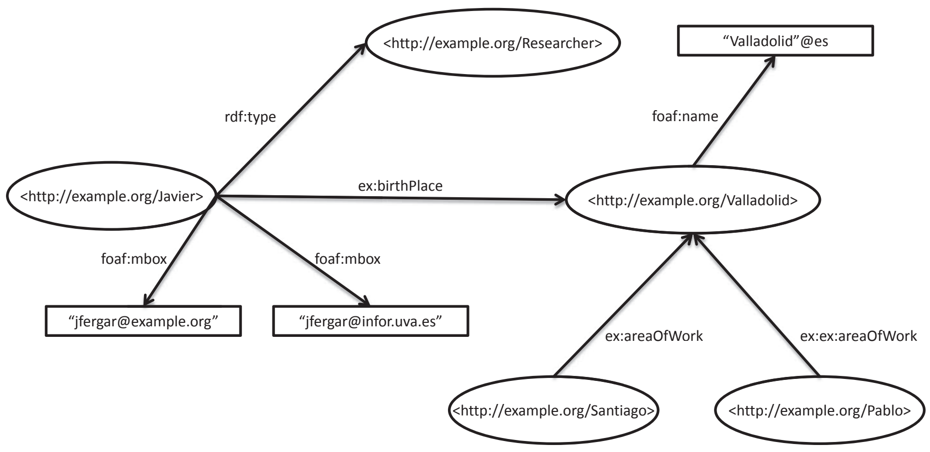
As stated, the subject out-degree indicates the cardinality of a subject node. In the example, the node *http://example.org/Javier* has a significant out-degree (it is related to four nodes, above average) and hence it conforms a star-shaped node. In practice, this type of nodes can have hundreds, or even thousands, of labeled edges.

When designing an RDF data structure, *e.g.* an index, it is potentially interesting to know the presence or absence of these nodes, but also the distribution of this high out-degrees. For instance, if a real-world RDF graph has a maximum out-degree close to 1, it stands for a very simple graph whose access may be optimized. In contrast, a skewed distribution of high out-degrees could require a more refined structure than the previous case.

Thus, out-degree distribution together with maximum and mean values constitutes a fair characterization 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.

Regarding partial and labeled out- and in- degrees, they provide 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 nodes categorization. For instance, in the example, *http://example.org/Valladolid* is a common object as three subjects are related to it, hence its in-degree is three. However, the labeled in-degree is “two” as it receives edges from two labels *ex:birthday* and *ex:areaOfWork*. Subsequently, its partial in-degree is two, denoting that the pair *(http://example.org/Valladolid, ex:areaOfWork)* is multivalued.

As we state in the forthcoming evaluation, labeled out-degree verifies that few predicates are related to the same subject or object. This could serve RDF structures to optimize the representation of the list of predicates related to a given subject or object.


<http://example.org/Researcher>
"Valladolid"@es
rdf:type
foaf:name
<http://example.org/Javier>
ex:birthPlace
<http://example.org/Valladolid>
foaf:mbox
foaf:mbox
"jfergar@example.org"
"jfergar@infor.uva.es"
ex:areaOfWork
ex:ex:areaOfWork
<http://example.org/Santiago>
<http://example.org/Pablo>

SUBJECT OUT-DEGREE	*Max*	*total*	$deg^-(G)$	4.00	OBJECT IN-DEGREE	*Max*	*total*	$deg^+(G)$	3.00
		partial	$deg^{--}(G)$	2.00			*partial*	$deg^{++}(G)$	2.00
		labeled	$deg^-_L(G)$	3.00			*labeled*	$deg^+_L(G)$	2.00
		direct	$deg^-_D(G)$	4.00			*direct*	$deg^+_D(G)$	3.00
	Mean	*total*	$\overline{deg^-}(G)$	1.75		*Mean*	*total*	$\overline{deg^+}(G)$	1.40
		partial	$\overline{deg^{--}}(G)$	1.17			*partial*	$\overline{deg^{++}}(G)$	1.17
		labeled	$\overline{deg^-_L}(G)$	1.50			*labeled*	$\overline{deg^+_L}(G)$	1.20
		direct	$\overline{deg^-_D}(G)$	1.75			*direct*	$\overline{deg^+_D}(G)$	1.40

PREDICATE DEGREE	*Max*	*total*	$deg_P(G)$	2.00	RATIOS	α_{s-o}	0.13
		out	$deg^-_P(G)$	2.00			
		in	$deg^+_P(G)$	2.00		α_{s-p}	0.00
	Mean	*total*	$\overline{deg_P}(G)$	1.40			
		out	$\overline{deg^-_P}(G)$	1.20			
		in	$\overline{deg^+_P}(G)$	1.20		α_{p-o}	0.00

Subject-Object degrees (restricted to <http://example.org/Valladolid>)

SUBJECT OUT-DEGREE (restricted to common s-o)	*Max*	*total*	$deg^-(G)\|_{s-o}$	1	OBJECT IN-DEGREE	*Max*	*total*	$deg^+(G)\|_{s-o}$	3
		partial	$deg^{--}(G)\|_{s-o}$	1			*partial*	$deg^{++}(G)\|_{s-o}$	2
		labeled	$deg^-_L(G)\|_{s-o}$	1			*labeled*	$deg^+_L(G)\|_{s-o}$	2
		direct	$deg^-_D(G)\|_{s-o}$	1			*direct*	$deg^+_D(G)\|_{s-o}$	3
	Mean	*total*	$\overline{deg^-}(G)\|_{s-o}$	1		*Mean*	*total*	$\overline{deg^+}(G)\|_{s-o}$	3
		partial	$\overline{deg^{--}}(G)\|_{s-o}$	1			*partial*	$\overline{deg^{++}}(G)\|_{s-o}$	1.5
		labeled	$\overline{deg^-_L}(G)\|_{s-o}$	1			*labeled*	$\overline{deg^+_L}(G)\|_{s-o}$	2
		direct	$\overline{deg^-_D}(G)\|_{s-o}$	1			*direct*	$\overline{deg^+_D}(G)\|_{s-o}$	3

Typed subjects (restricted to <http://example.org/Javier>) and classes (<http://example.org/Researcher>)

# Classes ($\|C_G\|$)= 1	SUBJECT OUT-DEGREE (restricted to typed S)	*Max*	*total*	$deg^-(G)\|_{s-o}$	4.00
			partial	$deg^{--}(G)\|_{s-o}$	2.00
# Typed Subjects ($\|S^C_G\|$)= 1			*labeled*	$deg^-_L(G)\|_{s-o}$	3.00
			direct	$deg^-_D(G)\|_{s-o}$	4.00
Ratio ($\frac{\|S^C_G\|}{\|S_G\|}$)= 25%		*Mean*	*total*	$\overline{deg^-}(G)\|_{s-o}$	4.00
			partial	$\overline{deg^{--}}(G)\|_{s-o}$	1.33
Max. lists per class ($deg_{LPC}(G)$)=1			*labeled*	$\overline{deg^-_L}(G)\|_{s-o}$	3.00
			direct	$\overline{deg^-_D}(G)\|_{s-o}$	4.00

Figure 4.1: Summary of structural metrics describing a small RDF graph example.

Finally, 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. In the example, direct degrees throw similar results as the out- and in-degrees, as every *(subject,object)* pair is related only with one predicate

Direct degrees could serve when representing RDF as a classical adjacency matrix. For instance, let us suppose that one builds a matrix in which rows represent subjects and columns represent objects. A marked cell, then, stands for a triple having the corresponding subject and object. The cell could include the predicate labels of the relationship. In such scenario, direct out-degrees model the cardinality of rows, whereas direct in-degrees describe the cardinality in columns. In turn, if all the cells hold a unique predicate, the out- and in- degrees of every subject and object are equivalent to the direct degrees.

4.1.2 Predicate Degrees

Despite the fact that important RDF characteristics can be extracted from the previous metrics (or a combination of them), one could argue that some RDF indexing techniques need further details. For instance the family of indexing techniques following vertical partitioning (Abadi, Adam, Madden, & Hollenbach, 2007) builds indexes per predicate (see a review in §6.2.1). Typically, these techniques index all the *(subject,object)* pairs for each predicate. In such scenario, the number of *(subject,object)* pairs for each predicate would be a good indicator of the size and distribution of these predicate partitions.

With this objective in mind, we detail predicate degrees following the same preceding principles of simplicity and use in other scenarios. The purpose of the metrics is summarized as follows:

- **predicate degrees:** to know the cardinality of predicates. In contrast to the relational model in which every row of a table is described with the same number of attributes (columns), the flexibility of RDF yields to a potentially high variability in the number of predicates describing each subject. Thus this metric is an important clue of the most important, or better said, most used, predicates in an RDF dataset.

- **predicate in- degrees:** to describe the number of subjects related to given predicates. It is used to refine the previous metric, specially useful when there are multivalued pairs *(subject,predicate)* heavily loaded which influence the previous metric.

- **predicate out- degrees:** to know the number of different objects related to given predicates, also used to describe the predicate degree in detail.

We make use of the aforementioned notation, being G an RDF graph, with S_G, P_G, O_G the sets of subjects, predicates and objects in G and generic $s \in S_G$, $p \in P_G$ and $o \in O_G$.

Definition 15 (predicate degree) *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*.*

Definition 16 (predicate in-degree) *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,* $deg_P^+(p) = |\{s \mid \exists z \in O_G, (s,p,z) \in G\}|$*. For the whole graph, the* maximum predicate in-degree*,* $deg_P^+(G) = max_{p \in P_G}(deg_P^+(p))$*, and its corresponding* mean predicate degree *value for graph* G*,* $\overline{deg_P^+}(G) = \frac{1}{|P_G|}\Sigma_{p \in P_G} deg_P^+(p)$*, are defined as the maximum and mean predicate in-degrees of all predicates of* G*.*

Definition 17 (predicate out-degree) *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,* $deg_P^-(p) = |\{o \mid \exists x \in S_G, (x,p,o) \in G\}|$*. For the whole graph, the* maximum predicate out-degree*,* $deg_P^-(G) = max_{p \in P_G}(deg_P^-(p))$*, and its corresponding* mean predicate out-degree *value for graph* G*,* $\overline{deg_P^-}(G) = \frac{1}{|P_G|}\Sigma_{p \in P_G} deg_P^-(p)$*, are defined as the maximum and mean predicate out-degrees of all predicates in* G*.*

Explanation and potential uses. As stated, the predicate degree constitutes an essential metric when a *(subject,object)* or *(object,subject)* is built for each predicate, such as the vertical partitioning technique (Abadi et al., 2007).

The 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).

For instance, if a predicate returns a high degree (it appears in many triples) but a low out-degree, it reveals that few values are repeated along descriptions. For instance, if we are describing individual records, this is the case of discrete values for predicates such as "City_State" or "Postal_code" in which a dozen of similar values could be repeated in thousands or millions of records.

In- and -out degree may also serve in other scenarios when individual subjects or objects for a given predicate must be indexed.

Figure 4.1 illustrates these metrics. Despite the limited size of the example, it shows the variable figures of predicate degrees. For instance, the predicate *foaf:name* is present only once whereas *foaf:mbox* and *ex:areaOfWork* are twice. In this latter, its predicate in-degree is two (denoting two different subjects) yet the out-degree is only one (all two subjects points to the same object). This example shows that predicate in- and out- degree could roughly classify predicate usage as follows:

- **N:N predicates.** These are predicates having a similar in- and out-degree, $deg^+_P(p) \simeq deg^-_P(p)$. Note that a special case would be *1:1 predicates*, *i.e.* predicates appearing only in one triple, but this is a marginal case at large scale[2].

- **1:N predicates.** These are predicates having a significant smaller in-degree than their out-degree, $deg^+_P(p) \ll deg^-_P(p)$.

- **N:1 predicates.** These are predicates having a significant greater in-degree than their out-degree, $deg^+_P(p) \gg deg^-_P(p)$.

Although the formal demonstration of this classification goes beyond the purpose of this thesis, one could envision that this is a general scenario in real-world datasets. For instance, predicates describing unique IDs, such as "Passport" or "Protein_ID", belong to *1:1 predicates*. In turn, the mentioned "City_State" or "Postal_code" fall into *N:1 predicates*. Finally, other predicates, such as "foaf:mbox" in the example, can belong to *1:N predicates*. Note that these examples seems perfect and clear examples, it could depend on the particular context and other predicates, though, can not be categorized beforehand such as "owl:sameAs" which depends on the concrete data.

4.1.3 Common Ratios

The presence of star nodes is popularly accepted as a natural consequence when describing a resource in depth. A second popular "construction" is the presence of chains, *i.e.*, paths of linked nodes. This construction occurs, for instance, whenever we use *owl:sameAs* to interlink two described entities. As some of these nodes in the chain is also a star, one could talk of "star chained design" for RDF datasets.

Intermediate nodes in chains appear in two triples acting with different roles. For instance, let us suppose a design such as $A \xrightarrow{p_1} B$ and $B \xrightarrow{p_2} C$. As shown, B is present in two triples, being an object in the first one, and subject in the latter. Additionally, we should also consider that predicates can again appear as nodes of other edges, acting also as intermediate nodes. In general terms, considering the three different roles in triples (subjects, predicates and objects), there could exist elements which are present in a graph acting with more than one role.

[2]It is generally accepted that the number of predicates is much smaller than the number of subjects and objects (Atre, Chaoji, Zaki, & Hendler, 2010).

We make use of three metrics to characterize the proportion of these common elements with respect to the total elements. In short:

- **subject-object ratio:** to describe the number of elements acting both as subject and objects among all subjects and objects. In other words, the subject-object ratio denotes the percentage of nodes having incoming and outgoing edges. They are, in fact, the main players when navigating the graph.
- **subject-predicate ratio:** to describe the number of elements acting both as subject and predicates among all subjects and predicates. Their presence points that semantics is given to predicates, *e.g.* using *rdfs:domain* or *rdfs:range*.
- **predicate-object ratio:** to describe the number of elements acting both as predicates and objects among all predicates and objects. It refines the previous metrics, *e.g.* when using *rdfs:subPropertyOf*.

Formally described, let us retake again G as an RDF graph, with S_G, P_G, O_G the sets of subjects.

Definition 18 (subject-object ratio α_{s-o}) *The subject-object ratio $\alpha_{s-o}(G)$ of a graph G is defined as the ratio of common subjects and objects in the graph G. Formally, $\alpha_{s-o}(G) = \frac{|S_G \cap O_G|}{|S_G \cup O_G|}$.*

Definition 19 (subject-predicate ratio α_{s-p}) *The subject-predicate ratio $\alpha_{s-p}(G)$ of a graph G is defined as the ratio of common subjects and predicates in the graph G. Formally, $\alpha_{s-p}(G) = \frac{|S_G \cap P_G|}{|S_G \cup P_G|}$.*

Definition 20 (predicate-object ratio α_{p-o}) *The predicate-object ratio $\alpha_{p-o}(G)$ of a graph G is defined as the ratio of common predicates and objects in the graph G. Formally, $\alpha_{p-o}(G) = \frac{|P_G \cap O_G|}{|P_G \cup O_G|}$.*

Explanation and potential uses. Ratios give evidence of chain constructions. The example in Figure 4.1 illustrates that there are no common subject-predicates and predicates-objects. In contrast, the subject-object ratio reveals that 13% of the subjects and objects are common elements which take part in a subject-object path.

Subject-object is, in fact, the most common construction as it is a natural way of linking the description of two resources. Subject-objects are key edges to index, because of the different roles they play, either as subjects described elsewhere, or as objects describing other resources. Thus, this ratio provides a good measure for data structures of the ratio of potential paths and the level of "navigability".

In turn subject-predicate and predicate-object ratios, when present, show how far predicates are also used as subjects or objects. These two ratios can be used to justify the consideration (or not) of a given RDF dataset as a graph. If there is a null influence of these types of shared nodes, one could assume that little semantics has been added.

4.1.4 Subject-Object Degrees

Given the importance of subject-object nodes, a fine-grained analysis can be made. In particular, one could study the in- and out-degrees restricted to subject-object nodes.

We define these degrees implicitly, as their formalization is equivalent to the degrees presented in Section 4.1.1, but restricted to subject-object nodes. For instance, the *maximum out-degree* of the graph G restricted to subject-objects, which is denoted as $deg^{-}(G)|_{s-o}$ is the maximum out-degree of all subject-object nodes in the graph G. That is, $deg^{-}(G)|_{s-o} = max_{s \in S_G \cap O_G}(deg^{-}(s))$. In the same way, the *mean out-degree* of the graph G restricted to subject-objects, $\overline{deg^{-}(G)|_{s-o}} = \frac{1}{|S_G \cap O_G|} \Sigma_{s \in S_G \cap O_G} deg^{-}(s)$, is defined as the mean out-degrees of all subjects-objects in G.

We make use of the same notation to define implicitly all out-degrees and in-degrees restricted to subject-objects, in Definitions 7 to 14.

For all the graph G:

- out- and in-degrees restricted to subject-objects: $deg^{-}(G)|_{s-o}$, $deg^{+}(G)|_{s-o}$ and their respective means $\overline{deg^{-}}(G)|_{s-o}$, and $\overline{deg^{+}}(G)|_{s-o}$.
- partial out- and in-degrees restricted to subject-objects: $deg^{--}(G)|_{s-o}$, $deg^{++}(G)|_{s-o}$, and their respective means $\overline{deg^{--}}(G)|_{s-o}$ and $\overline{deg^{++}}(G)|_{s-o}$.
- labeled out- and in-degrees restricted to subject-objects: $deg^{-}_{L}(G)|_{s-o}$, $deg^{+}_{L}(G)|_{s-o}$, and their respective means $\overline{deg^{-}_{L}}(G)|_{s-o}$ and $\overline{deg^{+}_{L}}(G)|_{s-o}$.
- direct out- and in-degrees restricted to subject-objects: $deg^{-}_{D}(G)|_{s-o}$, $deg^{+}_{D}(G)|_{s-o}$, and their respective means $\overline{deg^{-}_{D}}(G)|_{s-o}$ and $\overline{deg^{+}_{D}}(G)|_{s-o}$.

Explanation and potential uses. These metrics serve the same purposes as the original ones in Section 4.1.1, but restricted to subject-object nodes. This particularity allows to focus on these intermediate nodes and give a more detailed vision of what is going on in these important nodes.

Figure 4.1 provides these metrics over the given example. As only one subject-object node is present (*http://example.org/Valladolid*), the figures are simple: all out-degrees are equal to 1 because this node is solely related to the literal *"Valladolid"@es*. For in-degrees, the node playing the object role is presented in three triples with three different subjects.

This characterization might result specially useful when common subject-objects connects two different graphs. In such cases, one could grasp the features of these "connecting nodes" with these metrics, gaining insights to improve navigability. For instance, additional structures and indexes can be built for query suggestion or visualization purposes.

4.1.5 Predicate Lists

Subjects are described by means of one or more predicates. The list of predicates related to a subject may vary greatly for each subject. However, there would exist repetitions whenever two subjects are described in the same way. For instance, the list of predicates used to describe a *song* varies enormously from those used to categorize a *protein*, and both can coexist in a cross-domain dataset. We define metrics to characterize these lists. In short:

- **number and ratio of predicate lists**: it counts the number of different lists, and the ratio of lists from the total lists.
- **degree of predicate lists**: it characterizes the number of repetitions of each list.
- **lists per predicate**: it describes the number of different lists including each predicate.

Formally described, let L_s be the set of predicates (labels) related to the subject s. That is, the set of predicates $L_s = \{p \mid \exists z \in O_G, (s,p,z) \in G\}$. We denote as L_G to the set of different predicate lists in G. That is, $L_G = \{L_x, x \in S_G\}$, hence the **number of different lists** in the graph G is $|L_G|$. Note that the total predicate lists (with repetitions) is equal to the number of different subjects S_G.

Definition 21 (Ratio of repeated predicate lists) *The* ratio of repeated predicate lists $r_L(G)$ *of a graph G is defined as the ratio of repeated predicate lists from the total lists in the graph G. Thus, formally,* $r_L(G) = 1 - \frac{|L_G|}{|S_G|}$.

Definition 22 (predicate list degree) *The* predicate list degree *of a list L_s, denoted $deg_{PL}(L_s)$, is defined as the number of different subjects in G whose list of predicates is exactly L_s. Thus, formally, $deg_{PL}(L_s) = |\{L_x \mid x \in S_G, L_x = L_s\}|$. For the whole G, the* maximum predicate list degree, *$\underline{deg_{PL}}(G) = max_{L_x \in L_G}(deg_{PL}(L_x))$, and respectively the* mean predicate list degree *of the graph G, $\overline{deg_{PL}}(G) = \frac{1}{|L_G|}\Sigma_{L_x \in L_G} deg_{PL}(L_x)$, are defined as the maximum and mean out-degrees of all predicate lists in G.*

Definition 23 (lists per predicate degree) *The* lists per predicate degree *of a predicate p, $deg_{LPP}(p)$, is defined as the number of different predicate lists in L_G in which the predicate appears. Formally, $deg_{LPP}(p) = |\{L_x \mid p \in L_x, L_x \in L_G\}|$. For all G, the* maximum lists per predicate degree, *$\underline{deg_{LPP}}(G) = max_{p \in P_G}(deg_{LPP}(p))$, and respectively the* mean lists per predicate degree *of the graph, $\overline{deg_{LPP}}(G) = \frac{1}{|P_G|}\Sigma_{p \in P_G} deg_{LPP}(p)$, are defined as the maximum and mean out-degrees of all predicates in G.*

Explanation and potential uses. The presented metrics for the predicate lists characterize the repetition of predicates structures. On the one hand, if a short set of lists is present in all the entities, one could perfectly categorize this set and manage a reduce set of combinations. On the other hand, “random” lists denotes the presence of a cross-domain datasets or a light schema, as few repetitions are present.

The example in Figure 4.1, for instance, presents four predicate lists (one per subject): [*rdf:type*, *ex:birthPlace*, *foaf:mbox*], [*foaf:name*], [*ex:areaOfWork*] and again [*ex:areaOfWork*]. This latter is repeated in two different subjects, denoting a common structure (in spite of the reduced size of the example). In fact, the ratio of repeated predicate lists is $r_L(G) = 1 - \frac{3}{4} = 0.25$. This means that 25% of the predicate lists are repetitions. Note also that each predicate is present in only one list. In other words, in this particular case, predicates are unequivocally included in one list.

Predicate lists characterization would serve several purposes such as visualization or indexing. For instance, regarding the visualization scenario, the approach by Khatchadourian and Consens (2010) focuses in summarizing the links between Linked Open datasets. It is based on the notion of bisimulation contraction of a neighborhood (BCN), a structure which captures links between RDF datasets. In other words, BCN represents common predicate structures and modeling patterns of the original RDF graph. Our metrics may contribute to these summaries, as they categorize the type of repetitions.

In turn, regarding the indexing scenario, several approaches consider the commonalities in the predicate structures. Campinas, Perry, Ceccarelli, Delbru, and Tummarello (2012) make a structural summary grouping the entities having the same set of predicates in order to suggest potential predicates and relationships when writing a query. Tran, Ladwig, and Rudolph (2013) propose a structure index for RDF, grouping similar structured data elements. In both cases, the proposed metrics may help in determining structural properties of the indexes.

4.1.6 Typed Subjects and Classes

As stated, entities can be associated to types by means of the *rdf:type* predicate. The values for this predicate are then *Classes*, which can be described in detail by means of RDFS (see §2.1.1). For instance, in the example in Figure 4.1, *Javier* is of type *Researcher*. One should expect that, as previously mentioned, entities of the same class would be described with similar predicates. We define metrics to characterize these commonalities. In short:

- **number of classes**: it counts the number of different classes.
- **number and ratio of typed subject**: it counts the number of typed subjects (those including at least one type) and the ratio over the total subjects.
- **lists per class**: it describes the number of different predicate lists including each class.

- **out-degrees of typed subject**: it characterizes the out-degrees of typed subjects.
- **degree of predicate lists for typed subjects**: it characterizes the number of repetitions of those predicate list including at least one *rdf:type*.

Formally, let C_G be the set of all classes in the graph G, and c a generic class, $c \in C_G$. The **number of all different classes** is then $|C_G|$. Let S^c be the set of subjects of type c, $S^c = \{s \mid (s,t,c) \in G\}$, being t the predicate *rdf:type*. The set S_G^C denotes all different typed subjects in the graph G, that is $S_G^C = \{s \mid \exists c \in C_G, (s,t,c) \in G\}$, with $t =$ *rdf:type*. The number of different typed subjects in the graph is then $|S_G^C|$.

Definition 24 (Ratio of typed subjects) *The* ratio of typed subjects $r_T(G)$ *of a graph* G *is defined as the ratio of different typed subjects from the total subjects of* G*. Formally,* $r_T(G) = \frac{|S_G^C|}{|S_G|}$.

Let L_G^C be the set of different predicate lists for typed subjects. That is, $L_G^C = \{L_x, x \in S_G^C\}$.

Definition 25 (lists per class degree) *The* lists per class degree *of a class* c*,* $deg_{LPC}(c)$*, is defined as the number of different predicate lists in* L_G *in which the class* c *appears as a value for a typed subject. Formally,* $deg_{LPC}(c) = |\{L_x \mid L_x \in L_G^C, x \in S^c\}|$*. For all* G*, the* maximum lists per class degree, $\underline{deg_{LPC}}(G) = max_{c \in C_G}(deg_{LPC}(c))$*, and respectively the* mean lists per class degree *of the graph,* $\overline{deg_{LPC}}(G) = \frac{1}{|C_G|}\Sigma_{c \in C_G} deg_{LPC}(c)$*, are defined as the maximum and mean out-degrees of all classes in* G*.*

We define the *typed subject out-degrees* and the *degree of predicate lists for typed subjects* implicitly, as their formalization is straightforward. In the first case, the *typed subject out-degrees* are equivalent to those studied in Section 4.1.1, but restricted to typed subjects. For instance, the *maximum out-degree* of the graph G restricted to typed subjects, which is denoted as $deg^-(G)|_{S_G^C}$ is the maximum out-degree of all typed subjects in the graph G. That is, $deg^-(G)|_{S_G^C} = max_{s \in S_G^C}(deg^-(s))$. In the same way, the *mean out-degree* of the graph G restricted to typed subjects, $\overline{deg^-}(G)|_{S_G^C} = \frac{1}{|S_G^C|}\Sigma_{s \in S_G^C} deg^-(s)$, is defined as the mean out-degrees of all typed subjects in G. We make use of the same notation to define all out-degrees restricted to typed subjects, in the corresponding Definitions 7 to 10.

Next, the *degree of predicate lists for typed subjects* are equivalent to those studied in Section 4.1.5, but restricted to typed subjects. For instance, the *repetition ratio of predicate lists* restricted to typed subjects, $r_L(G)|_{S_G^C}$ of a graph G is defined as the ratio of different predicate lists from the total lists of predicates, both restricted to typed subjects. Formally, $r_L(G)|_{S_G^C} = \frac{|L_G^C|}{|S_G^C|}$. Similar reasoning can be made to define the *predicate list degree* of a list restricted to typed subjects, $deg_{PL}(L_s)|_{S_G^C}$, and the *lists per predicate degree* of a predicate restricted to typed subjects, $deg_{LPP}(p)|_{S_G^C}$.

Explanation and potential uses. The characterization of different classes and typed subjects, as well as their degrees, is an important step in describing a common schema, if present. As we have motivated, one should expect that subjects typed equally would be described with similar predicates. These metrics provide an answer to this assumption, and give insights of other schema features. For instance, the ratio of typed subjects constitutes a ratio of the level of well-categorized information. They also help determine if typed subjects are (or not) further described than non-typed ones.

Figure 4.1 (bottom) illustrates these metrics on the given example. There is only one class (*Researcher*) and one typed subject (*Javier*). As there are four different subjects, the ratio of typed subjects is 0.25. In this simple example, there is only one predicate list per class, [*rdf:type, ex:birthPlace, foaf:mbox*].

As for the previous predicate list metrics, this characterization may serve diverse purposes, such as visualization (Campinas et al., 2012) and structure indexing (Tran et al., 2013), but also reasoning. For this latter objective, we characterize not only the presence of instances for the classes, but the different predicate lists, which may be useful to create a reduced index with all the possible variants.

4.2 Experimental Framework

We design an experimental framework to illustrate the proposed metrics in real-world RDF datasets. Table 4.1 summarizes the most basic features of the experimental datasets.

First, in order to cover most real-world topics, we define **seven categories**: media, publications, knowledge base, government, sensors, geography and biology. We make this distinction based on the Linked Open data cloud most frequent topics[3].

We choose fourteen datasets based on the amount of triples, topic coverage, availability and, if possible, previous uses in benchmarking. Table 4.1 illustrates the datasets for each topic. Most datasets are well-known in the area. In particular:

- **Media:** *Jamendo* is a "small" dataset of music records and artists, *LinkedMDB* stores information about movies and authors, *Dbtune* provides music-related structured data (mainly from MySpace[4]), and *Flickr Event Media* (shorty known hereinafter as Flickr) holds Flickr events and their authors.
- **Publication:** *SWDF* is a small dataset with information related to the main conferences and workshops in the area of Semantic Web research, whereas *Faceted DBLP* (or DBLP hereinafter) is an RDF conversion of the well-known bibliographic repository.
- **Knowledge Bases:** *Wordnet 3.0* is a conversion to RDF of Wordnet (a lexical database of English) and *Dbpedia 3-8* is an RDF conversion of Wikipedia, with the aim of making this type of information semantically available on the Web.
- **Government:** The *2011 Australian Census* is an open portion of the given census with aggregated data and the *2000 US Census* comprises the first entities of the given census.
- **Sensors:** *AEMET* includes measurements made by the network of meteorological stations of the Spanish Meteorological Agency, and *Ike* contains meteorologic sensor information of the real Ike hurricane.
- **Geography:** *Linked Geo Data* holds geographic information mainly from the OpenStreetMap spatial data collection.
- **Biology:** *Affymetrix* contains probesets used in DNA microarrays.

A preprocessing phase is applied to all datasets. First, for a fair comparison, we manage all datasets in N-Triples (Grant & Beckett, 2004), one of the most basic formats containing one sentence per line. If the original dataset was not in N-Triples, it is converted to this raw format by means of the Any23 tool[5] (version: any23-0.6.1). Next, if the dataset is composed of several files, they are merged together. Finally, the dataset file is lexicographically sorted and duplicate triples are discarded.

Table 4.1 reflects the resulting figures after cleaning: the number of triples, the datasets size in N-Triples format, the given version of the dataset and the available URL.

Table 4.2 provides finer details of the datasets. The four latest columns show the number of different subjects, predicates, objects, and common subject-objects respectively. As expected, the number of

[3]We rename the topics from http://lod-cloud.net/state/

[4]Due to restrictions, we extract Dbtune information from the Billion Triples Challenge 2010 data collection.

[5]http://any23.apache.org/

	Dataset	Triples	Nt Size(MB)	Version	Available at
Media	Jamendo	1,049,637	144	2013-07-01	http://dbtune.org/jamendo
	LinkedMDB	6,147,996	850	2010-01-29	http://queens.db.toronto.edu/õktie/linkedmdb
	Dbtune	58,920,361	9,566	BTC 2010	http://km.aifb.kit.edu/projects/btc-2010
	Flickr Event Media	49,107,168	6,714	2010-07-01	http://www.eurecom.fr/ troncy/ldtc2010
Publications	SWDF	101,321	16	2013-07-01	http://data.semanticweb.org/dumps
	Faceted DBLP	60,139,734	9,799	2013-07-01	http://dblp.l3s.de/dblp++.php
Knowledge	Wordnet 3.0	6,257,922	974	2013-07-01	http://semanticweb.cs.vu.nl/lod/wn30
	Dbpedia 3-8	431,440,396	63,053	2013-07-01	http://wiki.dbpedia.org/Downloads38
Government	2011 Australian Census	361,842	52	2013-07-01	http://datalift.org/en/event/semstats2013/challenge
	2000 US Census	149,182,415	21,796	2007-08-14	http://www.rdfabout.com/demo/census
Sensors	AEMET	3,547,154	726	2011-11-19	http://aemet.linkeddata.es/source/rdf/data.zip
	Ike - Linked Observation Data	514,824,008	102,662	2013-07-01	http://wiki.knoesis.org/index.php/SSW_Datasets
Geography	Linked Geo Data	274,668,813	39,423	2013-07-01	http://downloads.linkedgeodata.org
Biology	Affymetrix	44,207,145	6,526	2012-11-06	http://download.bio2rdf.org/release/2/affymetrix

Table 4.1: Description of the evaluation framework.

	Dataset	Triples	#Subjects	#Predicates	#Objects	#Common SO
Media	Jamendo	1,049,637	335,925	26	440,602	290,291
	LinkedMDB	6,147,996	694,400	222	2,052,959	416,664
	Dbtune	58,920,361	12,401,228	394	14,264,221	10,076,199
	Flickr Event Media	49,107,168	5,490,007	23	15,041,664	3,822,727
Publications	SWDF	101,321	10,476	132	34,609	10,374
	Faceted DBLP	60,139,734	3,591,091	27	25,154,979	1,326,104
Knowledge	Wordnet 3.0	6,257,922	1,100,503	85	1,689,363	1,021,222
	Dbpedia 3-8	431,440,396	24,791,728	57,986	108,927,201	22,762,644
Government	2011 Australian Census	361,842	51,768	26	6,901	508
	2000 US Census	149,182,415	23,904,658	429	23,996,813	23,815,829
Sensors	AEMET	3,547,154	394,289	23	793,664	433
	Ike - Linked Observation Data	514,824,008	114,484,017	10	114,629,189	114,484,017
Geography	Linked Geo Data	274,668,813	51,916,995	18,272	121,749,861	41,471,798
Biology	Affymetrix	44,207,145	1,421,763	105	13,240,270	245

Table 4.2: Details of the evaluation framework.

predicates remains commonly low. There are two exceptions: *Dbpedia* and *Linked Geo Data* are extreme cases in which the number of predicates grows to the order of thousands due to the variability of the represented information. However, note that the number of predicates remains proportionally small to the total number of triples.

4.3 Results

We compute the parameters previously presented, in order to characterize the structure and gain insights toward the aforementioned potential uses. For our future purposes, we specially focus on analyzing the redundancy of each dataset, as well as their compact and compression possibilities.

For a comprehensive explanation, the order of presentation of the results is slightly different than the previous definitions.

4.3.1 Ratios

We start describing the common ratios in Table 4.3. These were described in Section 4.1.3, and they are a good starting point as they can reveal a level of cohesion between the different types of nodes. In other words, they can denote and characterize the presence (or absence) of shared nodes and labels.

As we expected, subject-object is the most frequent path constructor indeed and subject-predicate and predicate-object ratios are almost negligible. These latter are scheme descriptions, which are rare due to the RDF itself is schema-relaxed and the vocabulary can evolve as needed on demand.

	Dataset	RATIOS Common SO (α_{s-o})	Common SP (α_{s-p})	Common PO (α_{p-o})
Media	Jamendo	0.60	0	0
	LinkedMDB	0.18	0	1.66×10^{-5}
	Dbtune	0.61	0	3.44×10^{-6}
	Flickr	0.23	0	0
Publications	SWDF	0.30	0	0
	Faceted DBLP	0.05	7.52×10^{-6}	0
Knowledge	Wordnet 3.0	0.58	7.27×10^{-6}	1.78×10^{-6}
	Dbpedia 3-8	0.21	2.24×10^{-3}	7.50×10^{-5}
Government	2011 Australian Census	0.01	9.65×10^{-5}	8.67×10^{-4}
	2000 US Census	0.99	0	0
Sensors	AEMET	3.65×10^{-4}	0	0
	Ike	0.99	0	0
Geography	Linked Geo Data	0.31	0	4.52×10^{-7}
Biology	Affymetrix	1.67×10^{-5}	0	5.89×10^{-6}

Table 4.3: Ratios of the given Datasets .

The subject-object ratio shows interesting variable figures, ranging between 0 to 99%. Extreme cases are particularly of interest. For instance, the *2011 Australian Census* and *AEMET* present values near to 0 whereas their counterparts per category, the *2000 US Census* and *Ike* show values near of 99% of shared nodes. One can find the explanation in the diverse strategy followed to model the information. On the one hand, both the *2011 Australian Census* and *AEMET* describe particular values for a given entity (a statistic value or a sensor measure). Thus, a more "isolated" graph can be found in such cases where we represent certain measures. On the other hand, both the *2000 US Census* and *Ike* make use of intermediate nodes (blank nodes in the census and entity resources in Ike) to organize the different types of figures or measures.

The low subject-object ratio in *Faceted DBLP* and *Affymetrix* is due to a different reason. In both cases, the datasets describe entities with a high number of different literals values. In the first case, titles, identifiers, dates, homepages, etc., of authors, articles and conferences are scarcely repeated. In the second, *Affymetrix* also describes entities (probesets) by different literal values (for labels, identifiers, version, description, dates, etc.). In addition, although URIs are used as objects, they are further described (as subjects) in other different datasets in the *bio2rdf* project.

The rest of the datasets can be grouped into two categories: datasets holding around 20-30% of shared entities (*LinkedMDB*, *Flickr*, *SWDF*, *Dbpedia* and *Linked Geo Data*), or near 60% (*Jamendo*, *Dbtune* and *Wordnet*).

4.3.2 Out- and in-degrees

In this section we study the mean out- and in-degree for subjects and objects respectively. The mean results and their standard deviations are presented in Figure 4.2. For the sake of comprehensibility, we erase hereinafter those error bars which significantly exceed the range of the figure. In this case, all in-degree deviations are erased. It is worth mentioning that both axes are in logarithmic scale. We also plot a dashed line delimiting the 1 value.

As can be seen, most datasets present a limited mean number of triples per subject and object. Regarding the out-degree, its mean is modestly greater than 10 only for *DBLP*, *Dbpedia* and *Affymetrix*. This denotes that most datasets (even those with hundreds of millions of triples) present a mean of 10 triples per subject at most. In turn, the mean in-degree is even lower. All datasets apart from the *2011 Australian Census* have a lower mean in-degree than out-degree, being always smaller than 10. That is, given an object, it is present in a mean of 10 triples at most. The *2011 Australian Census* is a special case. A detailed analysis shows that it makes use of discrete values for most fields, hence these values are highly repeated in different subjects.

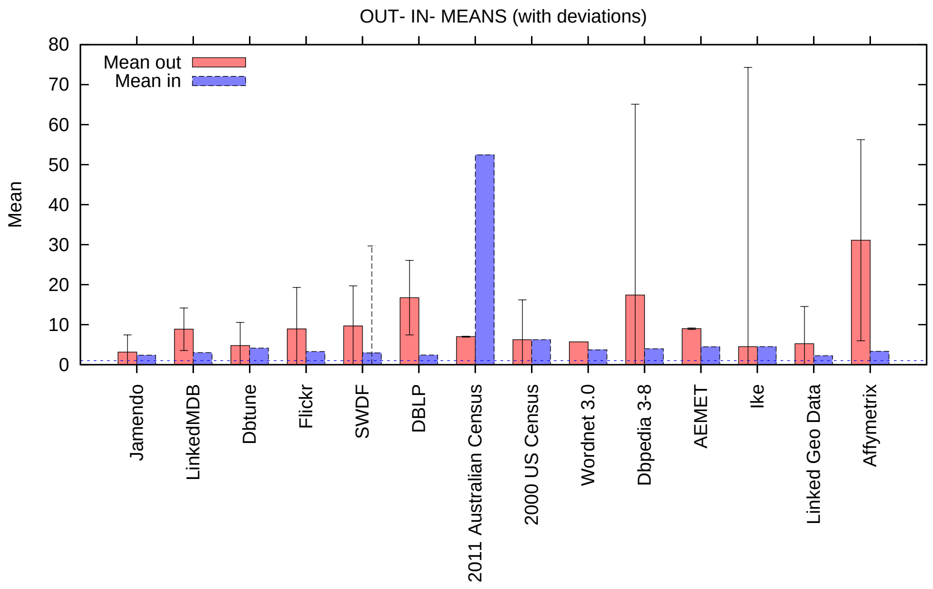


Figure 4.2: Mean out- and in-degrees for the evaluation datasets.

Both mean out- and in-degrees show, in general, a high standard deviation. In fact, all in-degree deviations exceed considerably the range of the figure. This points to a noticeable skewed structure, more remarkable in objects.

These skewed structures are revealed in Figures 4.3 to 4.6 which draw the out- and in-degrees. That is, we represent the cardinality of subjects and objects. We group datasets by categories for the only sake of clarity.

Several comments can be drawn from these figures. First of all, we can state that subjects and objects (out- and in-degree) almost always present skewed distributions. In fact, the in-degree in all datasets reveal a remarkably skewed structure on objects. Only two datasets, the *2011 Australian Census* (Figure 4.4, bottom left) and *AEMET* (Figure 4.6, top left) hold some objects slightly differing from the general tendency. In all the rest of cases, the distribution is heavily skewed.

In turn, subject distribution (out degree) is skewed in most datasets, but not all of them. The figures in our evaluation denote three types of patterns:

- Skewed distributions, as for objects. This is the case of all media datasets except for some blur in Jamendo (Figure 4.3), *SWDF* and *DBLP* (Figure 4.4), all knowledge datasets (Figure 4.5), *Linked Geo Data* and *Affymetrix* (Figure 4.6).

- A great number of different subjects are present in few triples. This can correspond to a structured data modeling in which subjects are described with a similar number of triples. We can find this circumstance in our two census (Figure 4.4, bottom left and right), and *AEMET* (Figure 4.6).

- A large number of different subjects are present in few triples, while many others are described with a small proportion of triples. Only *Ike* shows this type of distribution (Figure 4.6). It can be seen as a variation of the previous two types of distributions: some subjects are deeply described (or they have more relations) whereas others are concisely defined.

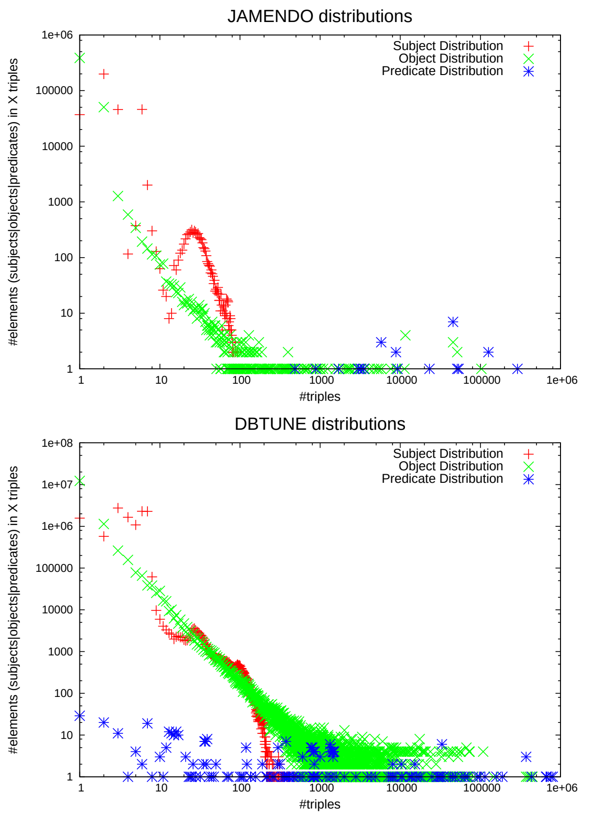


Figure 4.3: Degree distribution (media), in logarithmic scale.

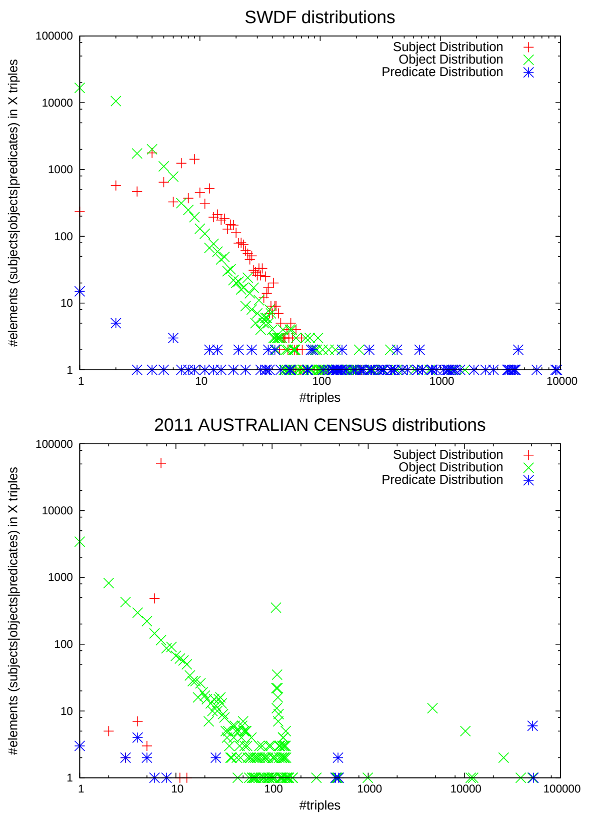


Figure 4.4: Degree distribution (publications and government), in logarithmic scale.

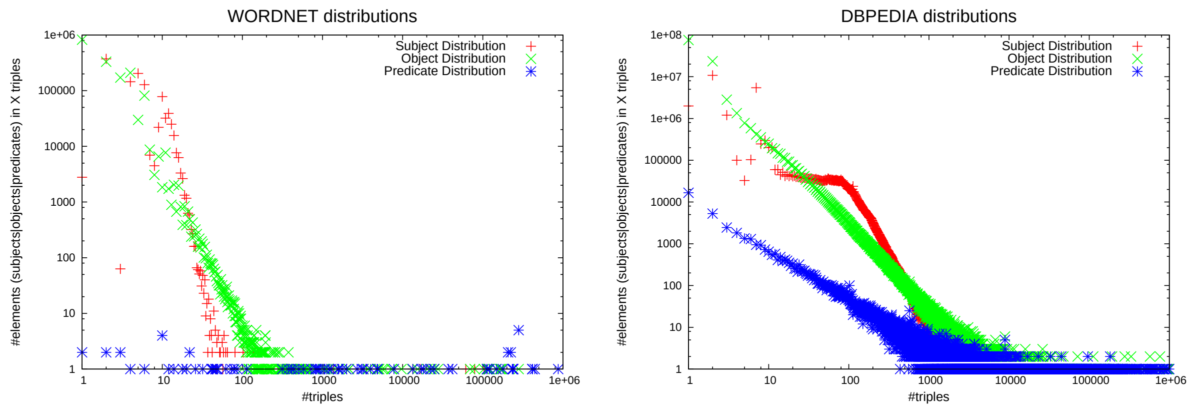


Figure 4.5: Degree distribution (knowledge), in logarithmic scale.

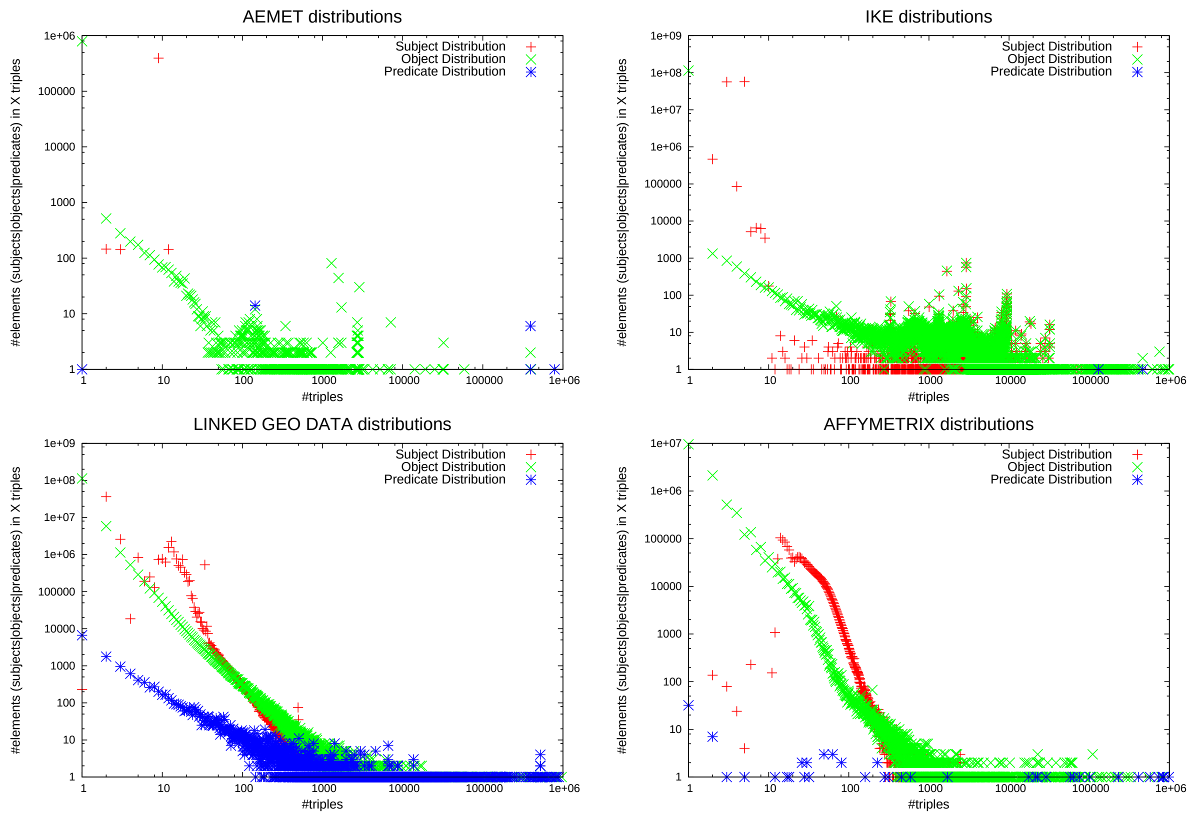


Figure 4.6: Degree distribution (sensors, geography and biology), in logarithmic scale.

Subject-object distribution. Figure 4.7 compares the previous mean out- and in-degree (presented in Figure 4.2) with the same degrees restricted to subject-object. For a fair comparison, we split the datasets by their range of common subject-object ratio (as stated in §4.3.1): common entities around 0%, 20-30%, 60% and 99%. We order the description of the results by these sets for explanation purposes:

- Common entities around 0%: In this case, the common entities are so rare that the means refer to few elements of the total. However, one could note that the mean in-degree restricted to these subject-objects is remarkably higher than for the total objects. We can find the reason of this

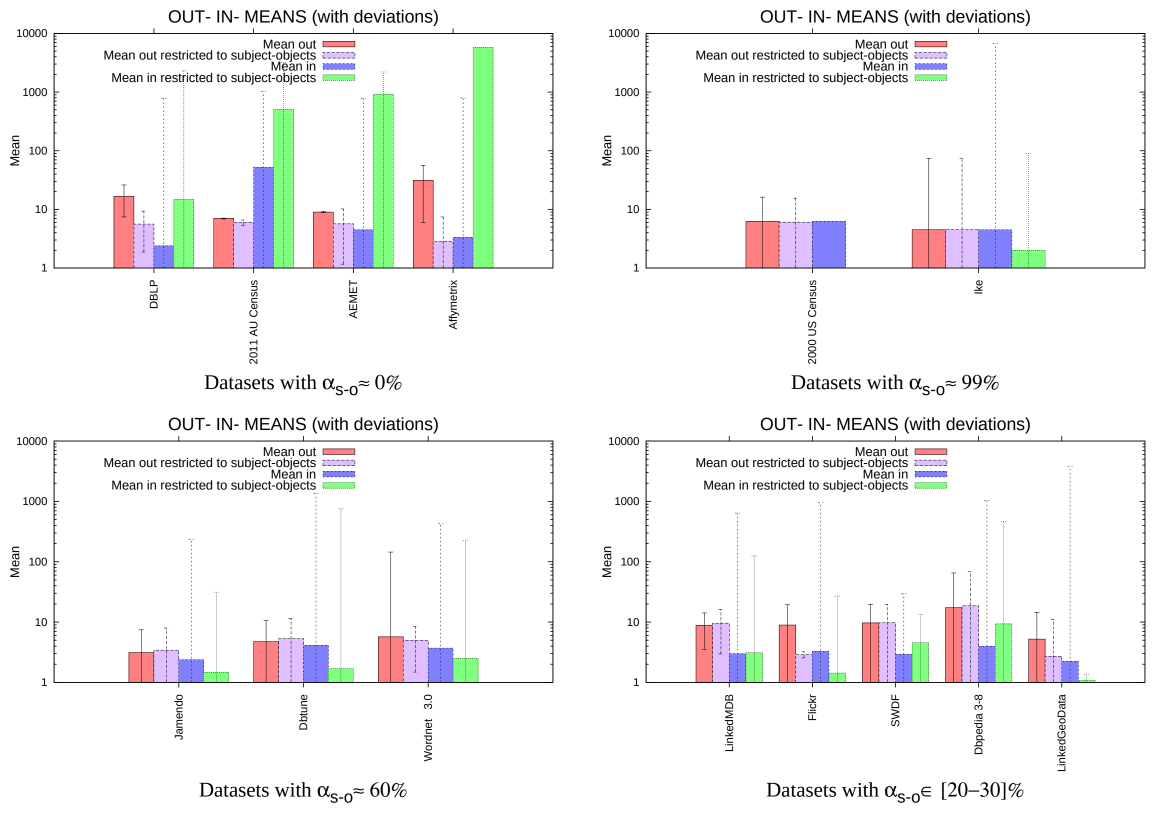


Figure 4.7: Mean out- and in-degrees for the evaluation datasets in comparison with the common subject-objects. The y-axis is represented in logarithmic scale.

difference in the non shared objects distribution. In all these datasets, a large number of different objects are present, whose in-degree is low (or even close to 1) as we can see in their corresponding in-degree distributions. Thus, the common subject-objects are more frequently present as they act as intermediate nodes and then playing as object in more triples on average.

- Common entities around 99%: This is the case of the *2000 US Census* and *Ike*. Figure 4.7 shows low figures for the mean in-degree, being exactly 1 for the *2000 US Census*. We have argued that both datasets make use of different shared elements to organize the different types of figures or measures, hence the low in-degree. In contrast, given that 99% of elements are shared, the mean out-degree for these nodes is almost equal to the out-degree for all subjects.

- Common entities around 60%: We can see that the mean out-degrees are almost equivalent as more than 50% of the elements are shared, hence these nodes highly contributes to the original figures. As for the previous case of common entities around 99%, this scenario shows low figures for the mean in-degree. The reason in this case is equivalent as intermediate nodes organize the information.

- Common entities around 20-30%: This is the most variable set and datasets can present different results. In general terms, the mean out-degrees remain comparable. Nevertheless, *Flickr* and *Linked Geo Data* show a slightly smaller out-degree for subject-object nodes. This fact clearly depends on the represented information. For instance, this phenomenon can appear when an "event" in *Flickr* is described in depth but the related subject-object nodes representing "authors" are usually described in lesser depth. Regarding the in-degrees, in some cases the figures restricted to subject-objects are equal, slightly smaller or bigger than the non restricted metric. The reasons are

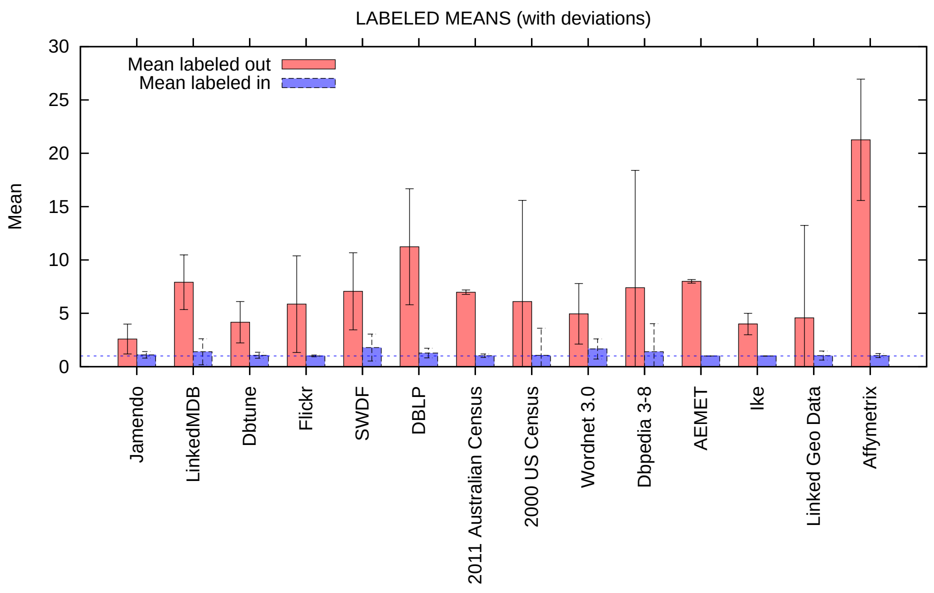


Figure 4.8: Mean labeled out- and in-degrees for the evaluation datasets.

similar to the presented above: it would be slightly smaller for subject-objects when they serve to organize the information and slightly bigger whenever non repeated objects are predominant.

4.3.3 Predicates per Subject and Object

We study the labeled out- and in-degrees, that is, the predicates per subject and object respectively. Figure 4.8 illustrates the mean figures. As can be seen, the results show that few predicates are related to the same subject, on average. *Affymetric* is the extreme case in which 20 predicates are present per subject. This fact, together with the mean out-degree (more than 30 triples per subject) reflects a description of entities in detail. In contrast, datasets such as *Jamendo* and *Ike* provide a mean of $3-4$ predicates per subject. In all cases, the mean labeled out-degree is a clear indicator of the presence of star-shaped nodes, *i.e.*, nodes with different triples around one common subject.

The mean labeled in-degree reveals an important conclusion. The number of predicates related to a given object is very close to 1. This stands for specific "leave nodes" reached by only one predicate.

The study of the maximum labeled out- and in-degrees, in Table 4.4 comes to similar conclusions. The results show that even in the extreme maximum cases, few predicates are related to the same subject and even less predicates per object. Besides, we provide in the table the ratio of maximum degrees over the total number of predicates. That is, a value of 20% for *Wordnet* means that, in the maximum case, a subject is related to the 20% of predicates in the dataset.

Finally, Figure 4.9 compares the mean labeled degrees of the common subject-objects with respect to the values obtained without restrictions. Two conclusions can be drawn form this comparison. First, the number of labels per common subject-object is generally equal or slightly smaller than the non restricted results. The corner case is *Affymetrix* which presents a significant reduction for subject-objects. One could argue that, in this case, general entities are detailed in depth whereas common subject-objects are simple nodes grouping discrete values and hence its smaller number of related predicates. Finally, it is important to note that the mean labeled in-degree of common subject-objects remains close to 1. This means that intermediate nodes (which are important for navigation as we have motivated) are reached by a mean of one unique predicate.

Dataset	Max. number of predicates per subject		Max. number of predicates per object	
	Labeled out deg. $(degL^{-}(G))$	Ratio $(\frac{degL^{-}(G)}{\lvert P_G \rvert})$	Labeled in deg. $(degL^{+}(G))$	Ratio $(\frac{degL^{+}(G)}{\lvert P_G \rvert})$
Jamendo	10	38.46%	5	19.23%
LinkedMDB	31	13.96%	50	22.52%
Dbtune	24	6.09%	93	23.60%
Flickr	14	60.87%	5	21.74%
SWDF	21	15.91%	13	9.85%
Faceted DBLP	18	66.67%	4	14.81%
Wordnet	17	20.00%	10	11.76%
Dbpedia 3-8	480	0.83%	6,005	10.36%
2011 Australian Census	7	26.92%	3	11.11%
2000 US Census	104	24.24%	366	85.31%
AEMET	12	52.17%	5	21.74%
Ike	5	41.67%	1	8.33%
Linked Geo Data	76	0.42%	3,431	18.78%
Affymetrix	35	33.33%	5	4.76%

Table 4.4: Values and ratios of the maximum labeled out- and in-degree for the experimental framework.

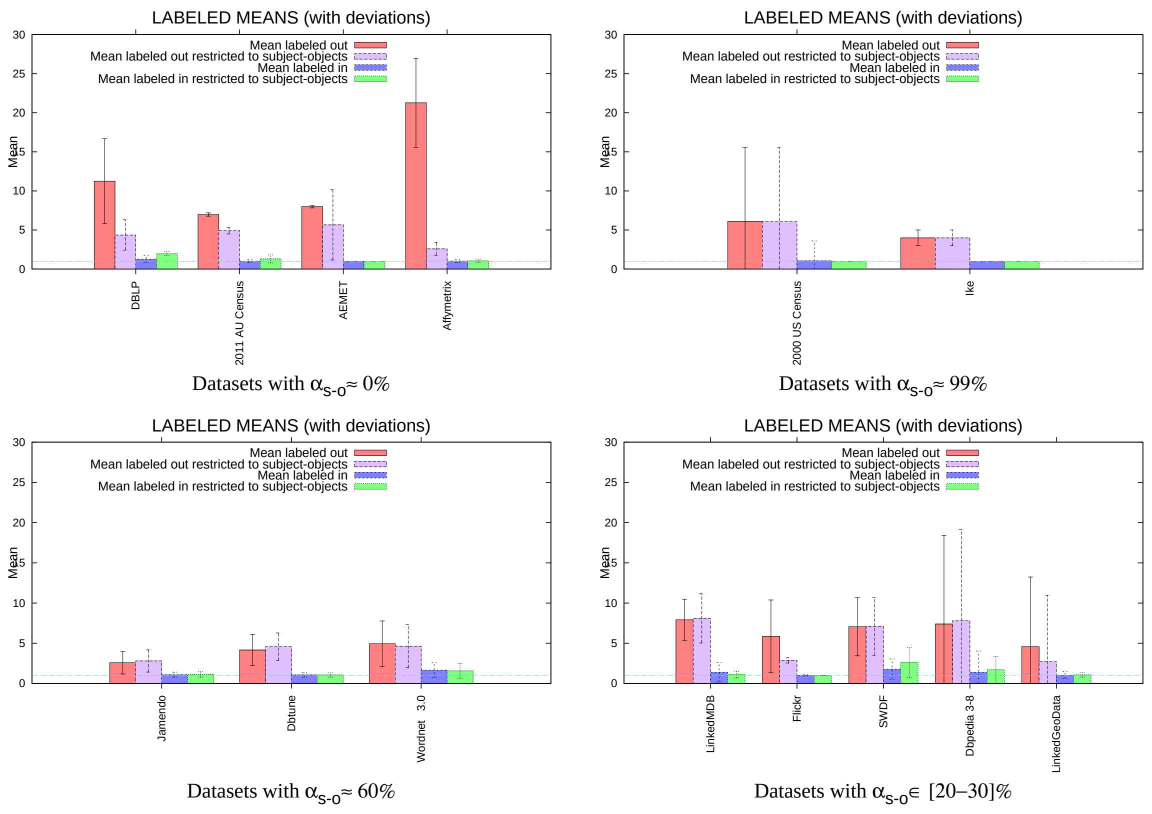


Figure 4.9: Mean labeled out- and in-degrees of common subject-objects for the evaluation datasets with respect to the values obtained without restrictions.

4.3.4 Partial and Direct Degrees

Figure 4.10 shows the mean partial out- and in-degrees. First of all, let us remember that partial out- and in-degrees reflect the presence of multivalued $(subject, predicate)$ and $(predicate, object)$ pairs respectively. As we can see, the mean partial out-degree is slightly bigger than 1, which implies that the presence of multivalued $(subject, predicate)$ pairs is not so frequent. In fact, the deviation is not pronounced (except for *Wordnet*) which denotes a uniform distribution.

In contrast, the mean in-degree remains close to 1, but it presents bigger deviations. Almost all deviation extends the range of the figure and they have been erased for the sake of clarity. This fact

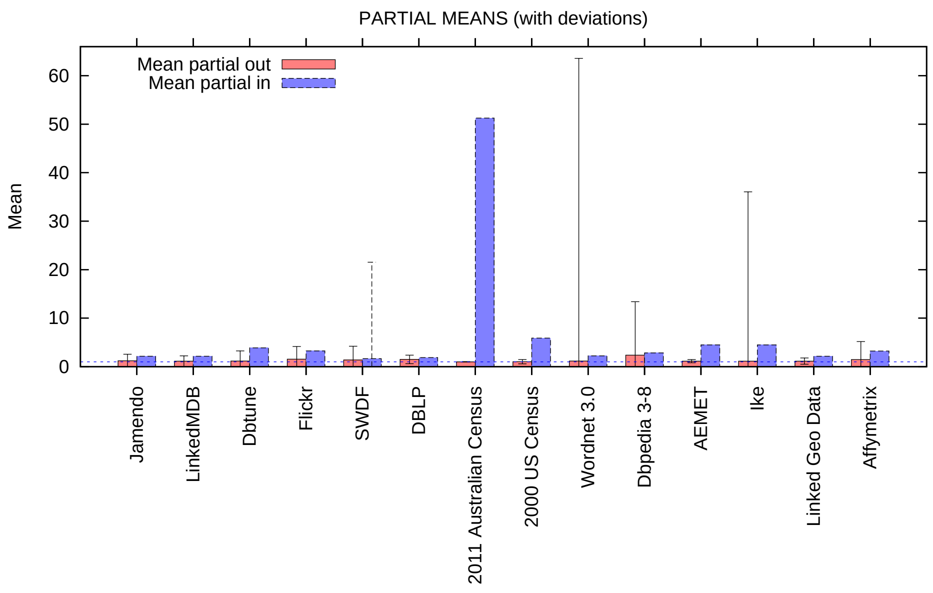


Figure 4.10: Mean partial out- and in-degrees for the evaluation datasets.

denotes a pronounced skewed distribution of multivalued $(predicate, object)$ pairs. That is, a large amount of different subjects are related to the same $(predicate, object)$ (*e.g.* this can be the case of rdf:type and its related classes) while others pairs are related to few different subjects, being 1 on average.

Next, we study the direct degrees, which measure the relationship between subjects and objects disregarding the presence of predicates. We have stated that the *direct out-degree* of a subject (the number of different related objects) can only differs from the *out-degree* (different related triples) when the subject is related to the same object by means of more than one predicate. Analogously, the *direct in-degree* of an object differs from the *in-degree* when the object is related to the same subject by means of more than one predicate.

We represent in Figure 4.11 the comparison between the mean out- and in-degrees and their respective mean direct degrees. The results show that the out-degree and the direct out-degree have similar figures, and the same applies to in-degree and direct in-degree. That is, results yield to an important state: given a subject and an object, if they are related, only one predicate brings these nodes together, on average.

Another remarkable fact is the difference, in some cases, between the mean direct out- and in-degrees. For instance, in *Dbpedia*, subjects are related with 5 times more objects than vice versa, reaching to 10 times in *Affymetrix*. This corresponds with datasets in which, on average, a subject is described in depth with different objects, which are no heavily repeated between subjects. In contrast, objects are related with 8 times more subjects than vice versa in the *2011 Australian Census*. This is due to large repetitions of the same set of objects in multiple subjects.

4.3.5 Predicate Degrees

In this section we study the predicate degrees, that is, the cardinality of predicates. We also detail their out- and in-degrees, which stands for the different objects and subjects related to each predicate.

First, Figure 4.12 shows the mean predicate degrees for all datasets. Note that this mean is highly biased by the number of triples of each dataset (figures are represented in logarithmic scale). For instance, conserving the same modeling, one could add other observations for the hurricane *Ike*, and the mean cardinality of the predicates will be increased.

In general terms, we can observe that the mean predicate out degree is slightly smaller than the

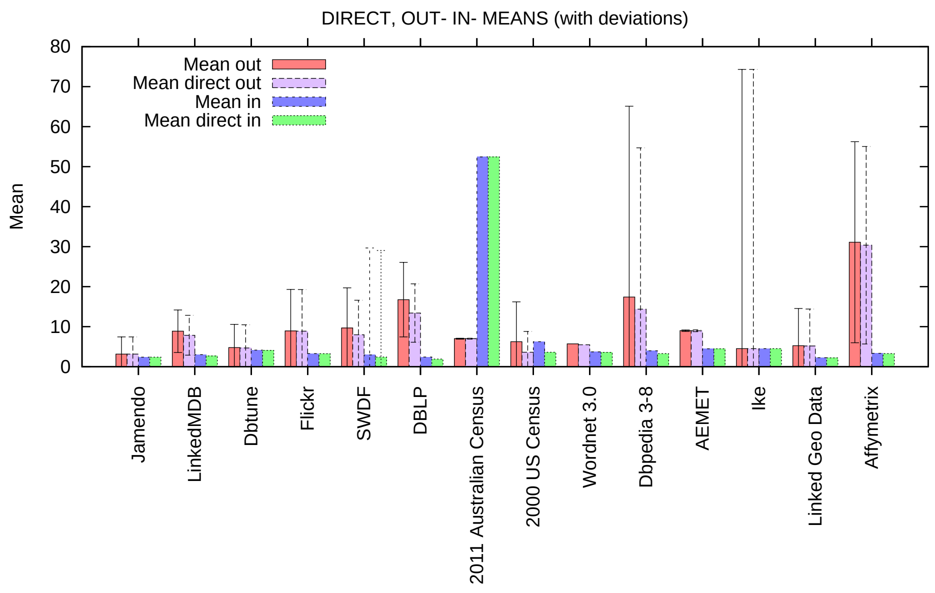


Figure 4.11: Mean direct degrees in comparison with mean out- and in-degrees for the evaluation datasets.

corresponding mean in degree. That is, given a predicate at random, it is probably related with more subjects than objects. This fact is in line with previous labeled and partial measurements; subjects are more related to predicates than objects, and multivalued $(subject, predicate)$ pairs are, when present, more infrequent than $(predicate, object)$ pairs.

We study in the following the distribution of predicates, as they can reveal different use patterns for the predicates. Figures 4.13 to 4.15 illustrate the degree of each predicate as well as their out- and in-degrees. It is clear that no prior assumption can be made on predicate distribution. In general terms, predicates distribution is tight to the information modeling. We can roughly distinguish three types of patterns in predicates:

- Mostly all predicates are present in every entity. In this case we could find a distribution such as the presented in *Flickr* (Figure 4.3, bottom right) or the *2000 US Census* (Figure 4.4, bottom right). In such case, the predicates are in the same region as they participate in a similar range of triples.

- Some predicates are present rarely while others are frequently used. This is the case of several representations such as all media datasets except for *Flickr* (Figure 4.3), all from publications and government except for the *2000 US Census* (Figure 4.4), *Wordnet* (Figure 4.5, left), and all from sensors, geography and biology except for *Linked Geo Data* (Figure 4.6).

- It is common that predicates are present in a reduced number of triples, whereas few predicates are related to thousand or millions of triples. This corresponds to the definition of a power law distribution. We can find this very clear skewed distributions in cross-domain datasets such as *Dbpedia* (Figure 4.5, right), or datasets including information about a given domain but mixed from diverse sources such as *Linked Geo Data* (Figure 4.6, bottom left). Due to the same reasons, these two datasets hold the higher numbers of predicates of all evaluation datasets.

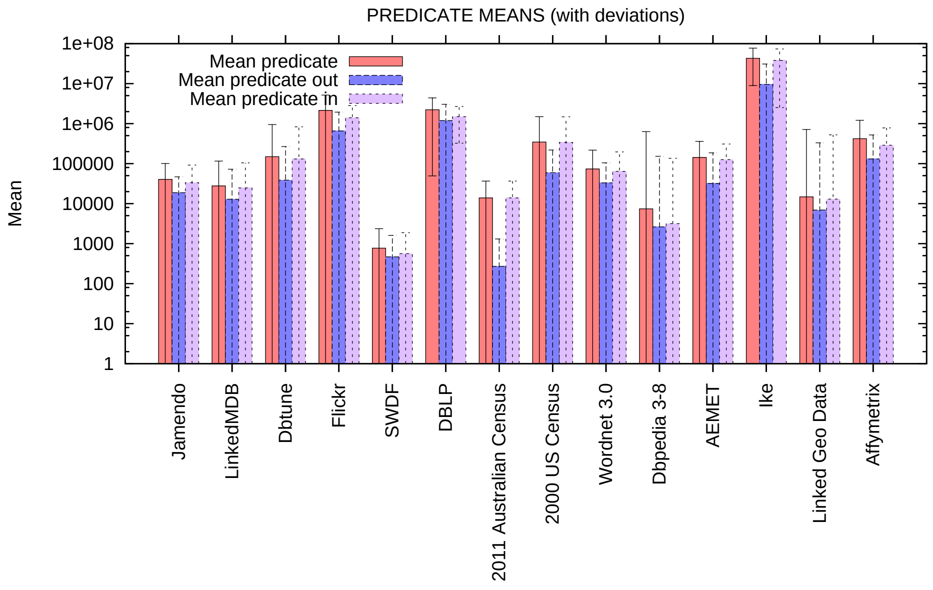


Figure 4.12: Mean predicate degrees for the evaluation datasets. The y-axis is in logarithmic scale.

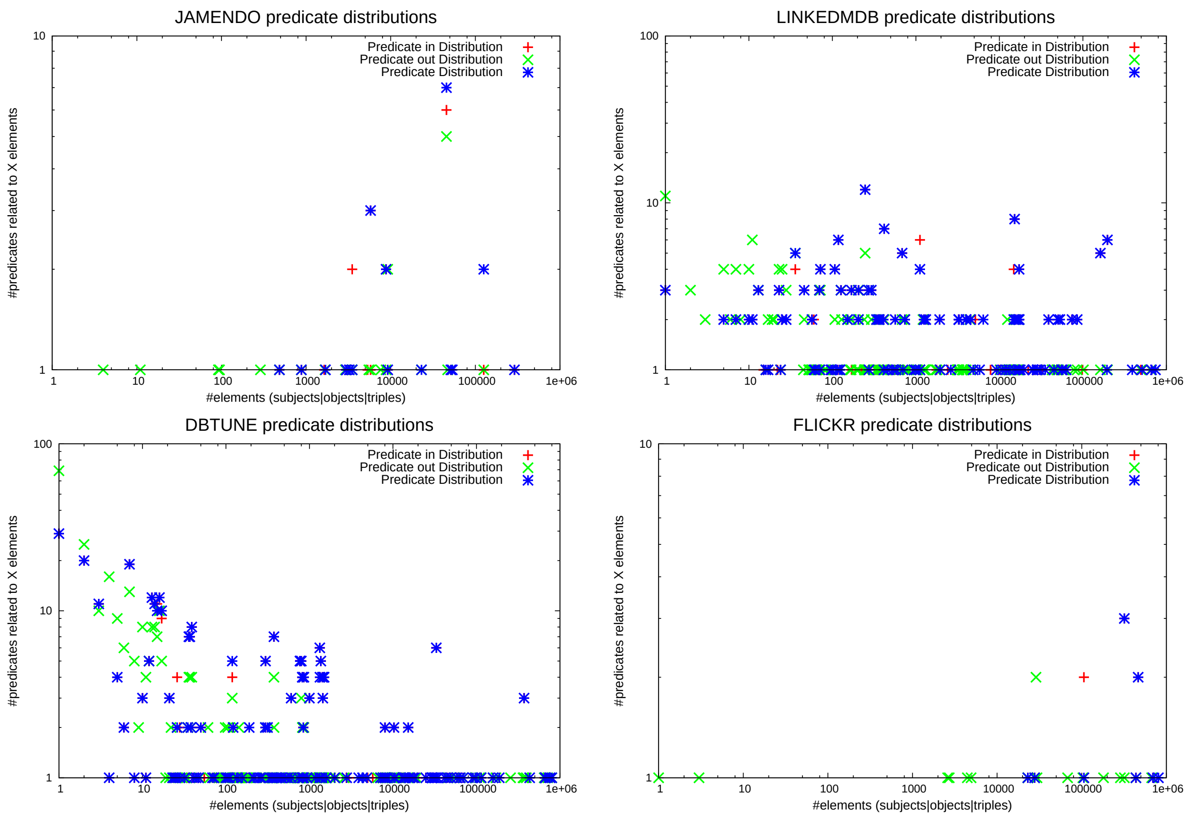


Figure 4.13: Predicate degree distribution (media), in logarithmic scale.

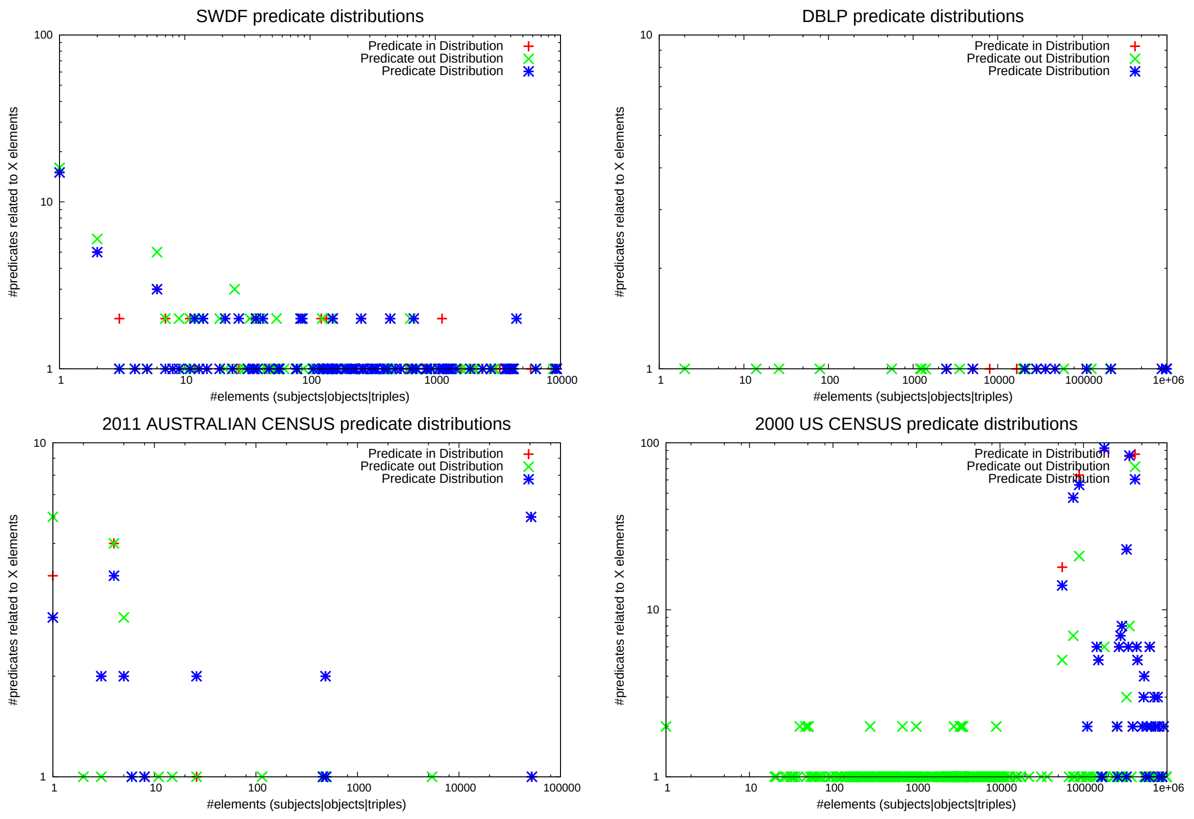


Figure 4.14: Predicate degree distribution (publications and government), in logarithmic scale.

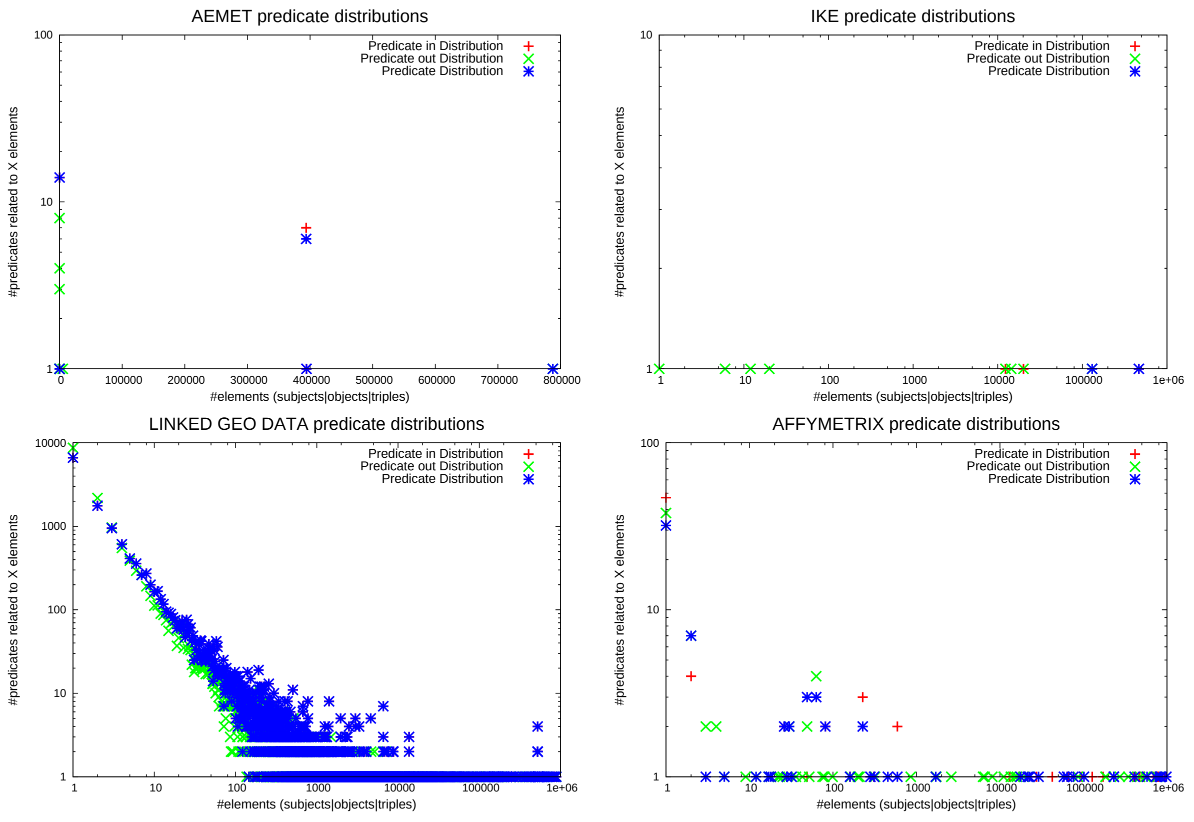
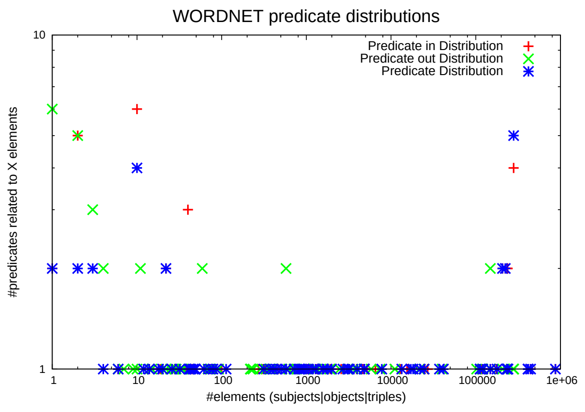


Figure 4.15: Predicate degree distribution (sensors, geography and biology), in logarithmic scale.



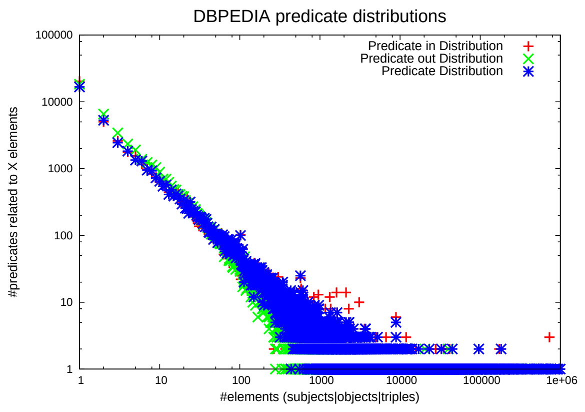


Figure 4.16: Predicate degree distribution (knowledge).

4.3.6 Study of Predicate Lists

We perform a study of the different predicate lists and their distribution. As we have motivated, there would exist repetitions whenever several subjects are described in the same way. Our goal is to establish to what extent these lists are repeated.

Table 4.5 (left) presents the number of different predicate lists and the repetition ratio. As can be seen, the number of different predicate lists is spectacularly low in all cases. For instance, *Jamendo* holds 26 predicates (as shown in Table 4.2), and between all potential combinations, only 43 different lists are present ($999.872‰$ of the lists are repetitions). It is also significant in the *2000 US Census* in which only 106 different lists appear from 429 predicates ($999.996‰$ of the lists are repetitions). Regarding those cross-domain datasets with more predicates and thus different entities, the proportion remains over $947‰$ repetitions. This is the case of Dbpedia, and it is also valid for *Linked Geo Data*.

Table 4.2 (right) shows the results of these metrics restricted to typed subjects (which are discussed in Section 4.3.7). As can be seen, predicate lists for typed subjects (Table 4.2, right) behaves similarly to the general case. The bigger difference is present in Dbpedia, in which the proportion of repeated lists decreases up to $712.413‰$. Nevertheless it remains significantly high once we are describing different type of entities, and we still found massive repetitions.

Thus, one can state that, in general terms, predicate lists are massively repeated. Next, we study the number of repetitions per list on average, and their distribution. This mean has been defined as the mean predicate list degree (Definition 22), and the results are shown in Figure 4.17.

These results are in line with the presented repetition ratio. Nevertheless, it is important to note that, as for the predicate cardinality, these results can be highly biased by the number of triples (the y-axis is in logarithmic scale). For a fairer characterization, we study the distribution of these repetitions in Figures 4.18 to 4.21.

One could expect that these distributions would correspond to the predicate distributions presented in the previous Section 4.3.5. That is, if a skewed distribution is present in predicates, the same result could be found in predicate lists. In contrast, if all predicates participates in similar number of triples (uniform distribution), the same shape is shared in predicate lists. Current results denotes that both assumptions remain true, with some interesting remarks described below. We consider the same previous categorization of predicates:

- Mostly all predicates are present in every entity, such as *Flickr* (Figure 4.18, bottom right) and the *2000 US Census* (Figure 4.20, bottom right). In such case, a similar non skewed distribution is present. Predicate lists are highly repeated, although they do not have to share the same number of repetitions.

Dataset	ALL subjects		TYPED subjects	
	# Dif. pred. lists ($\lvert L_G \rvert$)	Repetition ratio ($1 - \frac{\lvert L_G \rvert}{\lvert S_G \rvert}$)	# Dif. pred. lists ($\lvert L_G^C \rvert$)	Repetition ratio ($1 - \frac{\lvert L_G^C \rvert}{\lvert S_G^C \rvert}$)
Jamendo	43	999.872‰	41	999.859‰
LinkedMDB	8,459	987.818‰	8,442	987.314‰
Dbtune	963	999.922‰	782	999.922‰
Flickr	25	999.996‰	22	999.987‰
SWDF	364	965.254‰	341	961.754‰
Faceted DBLP	254	999.929‰	254	999.929‰
Wordnet	872	999.208‰	868	999.007‰
Dbpedia 3-8	1,309,392	947.184‰	1,152,617	712.413‰
2011 Australian Census	14	999.730‰	14	999.730‰
2000 US Census	106	999.996‰	-	-
AEMET	5	999.987‰	5	999.987‰
Ike	5	1,000.000‰	4	1,000.000‰
Linked Geo Data	220,902	995.745‰	219,015	995.562‰
Affymetrix	9,434	993.365‰	9,424	993.369‰

Table 4.5: Number and ratio of predicate lists for all subjects (left) and restricted to typed subjects (right).

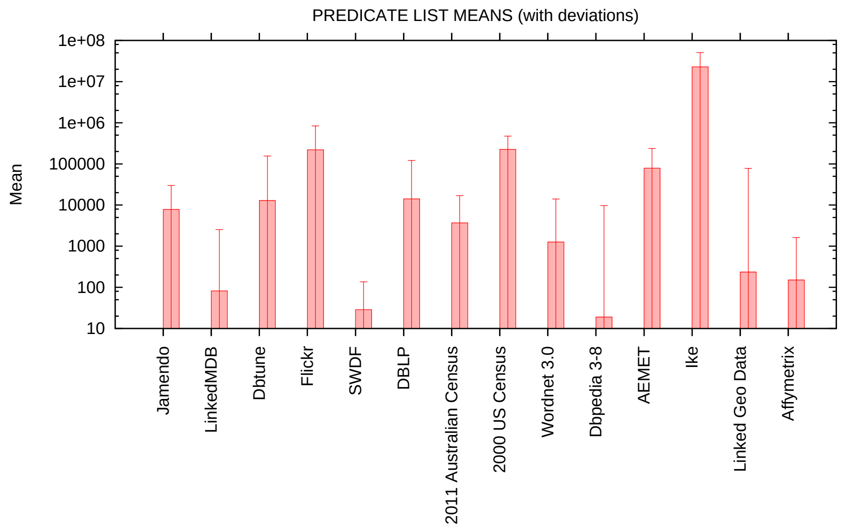


Figure 4.17: Mean predicate list degree for the evaluation datasets, in logarithmic scale.

- Some predicates are present rarely while others are frequently used, such as all media datasets except for *Flickr* (Figure 4.18), all from publications and government except for the *2000 US Census* (Figure 4.20), *Wordnet* (Figure 4.19, left), and all from sensors, geography and biology except for *Linked Geo Data* (Figure 4.21, bottom left). This is the most variable pattern. In fact, some of these datasets evolve to skewed distribution of predicate lists while others are not so marked. One can state that, whenever a slight skewed distribution is present in predicate degree, this evolves to a marked skewed distribution (power law) in predicate lists. Compare, for instance, the distribution of lists of *Affymetrix* (Figure 4.21, bottom right) with its predicate degree distribution (Figure 4.15, bottom right). This is the case of *LinkedMDB* and *DBTUNE* (Figure 4.18), *SWDF* and *DBLP* (Figure 4.20) and *Wordnet* (Figure 4.19), and the aforementioned *Affymetrix* (Figure 4.21).

  Other skewed structures different than power law distributions can be present. This is the case of *Jamendo* (Figure 4.18, top left) and the *2011 Australian Census* (Figure 4.20, bottom left).

- It is common that predicates are present in few triples, whereas others are related to thousand or millions of triples. This also evolves to clear skewed distributions (power law) of predicate lists in datasets such as *Dbpedia* (Figure 4.19, right), and *Linked Geo Data* (Figure 4.21, bottom left). Note that, due to the same reasons, these two datasets hold the highest numbers of predicates of all evaluation datasets.

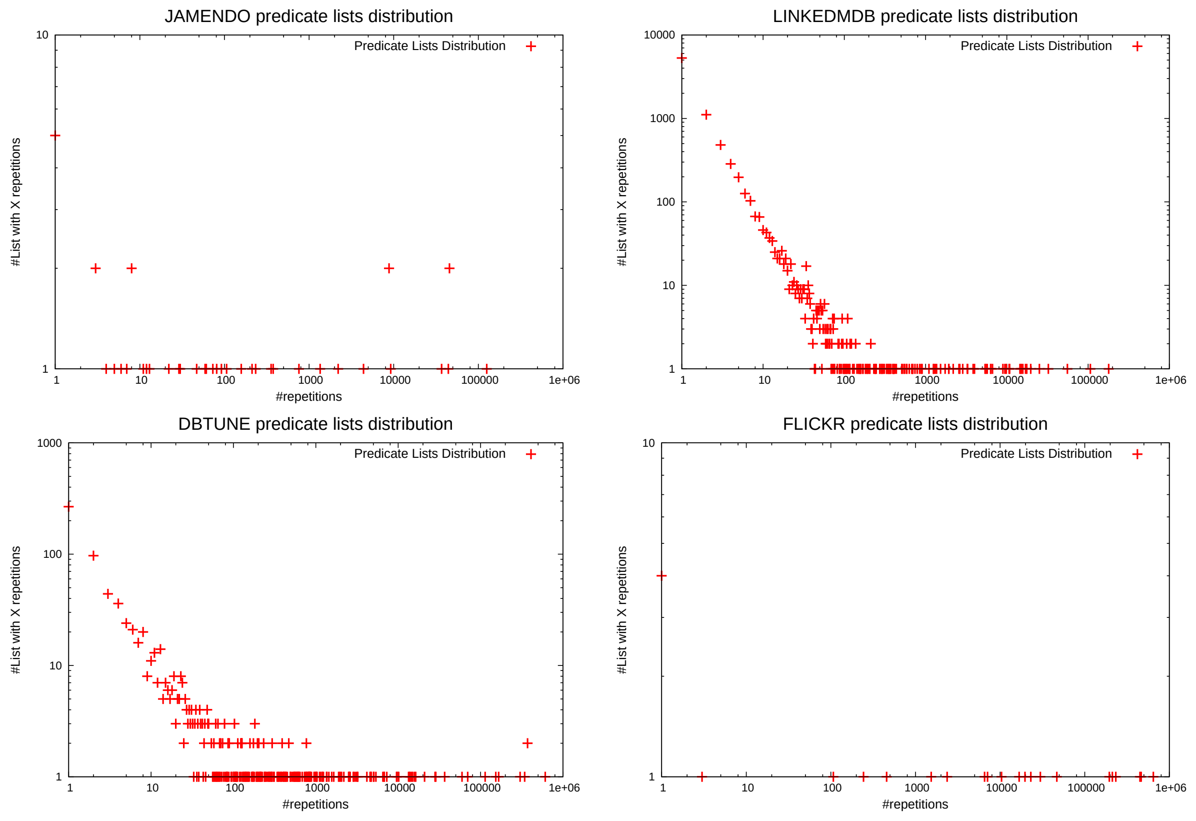


Figure 4.18: Predicate list degree distribution (media), in logarithmic scale.

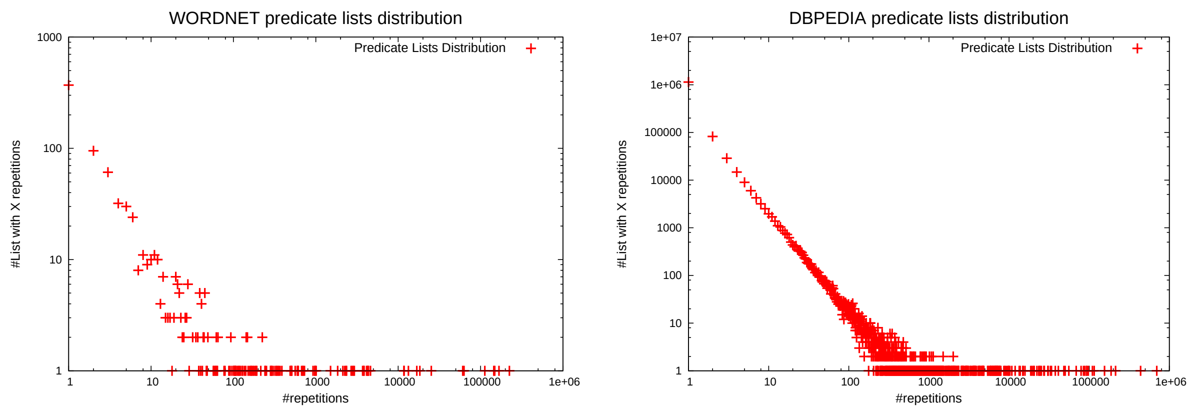


Figure 4.19: Predicate list degree distribution (knowledge), in logarithmic scale.

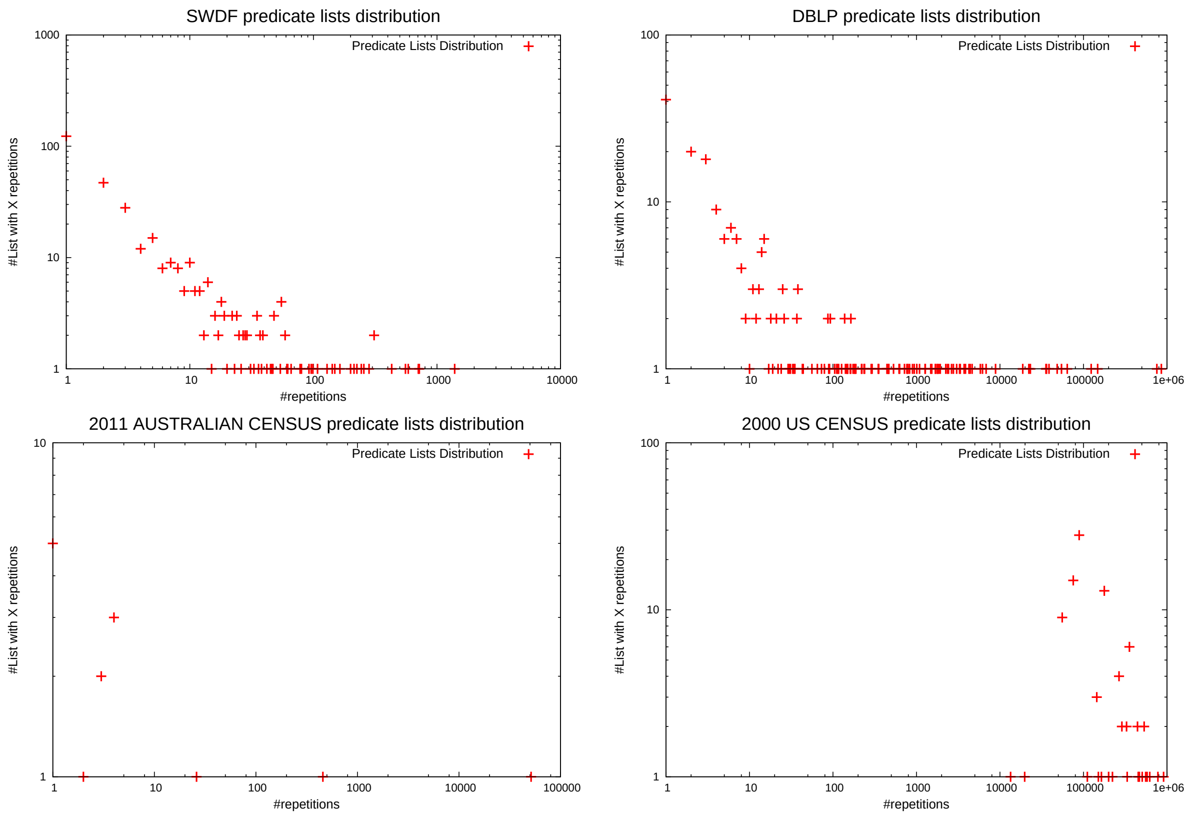


Figure 4.20: Predicate list degree distribution (publications and government), in logarithmic scale.

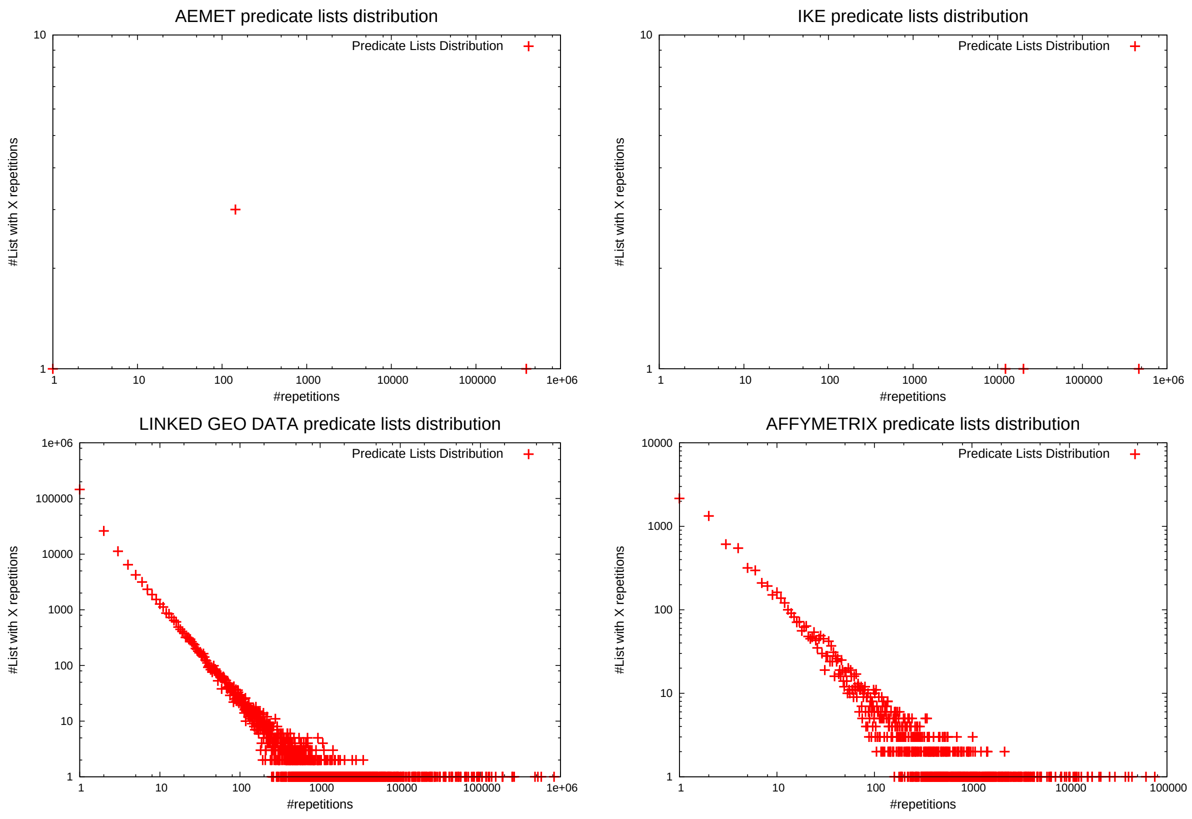


Figure 4.21: Predicate list degree distribution (sensors, geography and biology), in logarithmic scale.

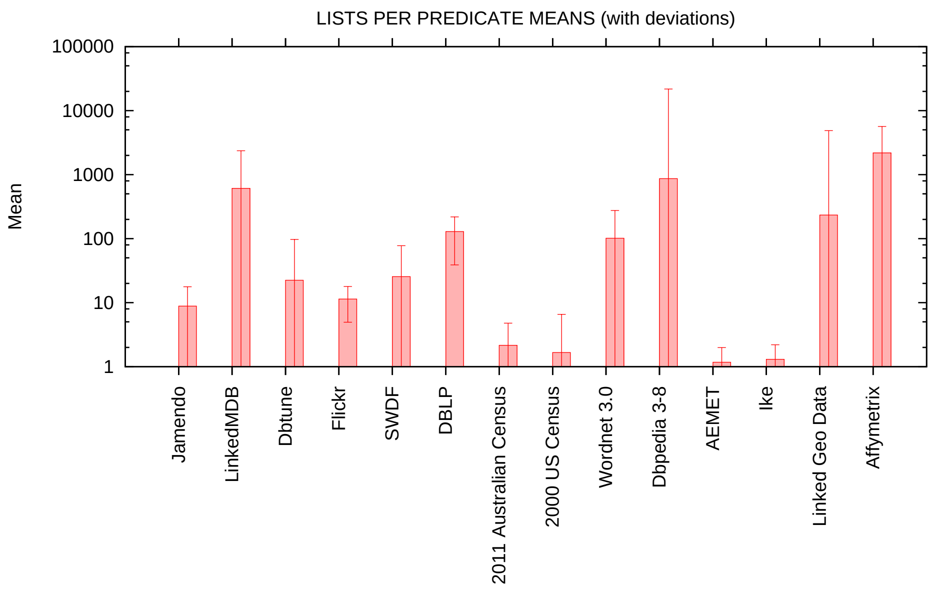


Figure 4.22: Mean list per predicate degree for the evaluation datasets, in logarithmic scale.

Finally, we study the number of different lists per predicate, on average. This is shown in Figure 4.22 (in logarithmic scale) which shows a significant low number of different lists in which a predicate is present. Note that if a predicate was related to one or two lists, given a predicate it is almost direct to know its peer predicates for any subject or object, even for the biggest datasets. In other words, the nearer this mean is to 1, the easier could be to discern the concrete list given a predicate, even in the biggest datasets.

The highest figures are obviously obtained for those datasets with more different datasets, but they remain proportionally small to the number of lists.

4.3.7 Study of Classes and Typed Subjects

We finish our evaluation with a brief study on typed entities. As we have argued, the following chapters of this thesis will consider all predicates regardless of the distinction between classes. Nevertheless, we incorporate this evaluation given the importance for other uses such as reasoning (see the potential uses of these metrics in Section 4.1.6).

Table 4.6 shows the resulting number of classes, typed subjects and the ratio of these typed subjects over the total subjects. Several remarks should be considered. First of all, one could expect that the larger is the dataset, the more classes are included. However it is worth remembering that RDF holds a relaxed schema, hence this assumption can result completely false. In other words, a “small” dataset such as *Jamendo* or *SWDF* can include more classes than the bigger *Flickr* or *AEMET*. Thus, the number of classes and typed subjects is completely biased by the data modeling and the domain, or domains, involved in the dataset.

With this assumption in mind, we can find in the results that the number of classes remain proportionally small with respect of the number of triples and entities. This is an obvious result as classes model common semantic types of entities, and this distinction should be limited. However, the ratio of typed subjects draws more interesting results. Table 4.6 reflects three types of modelings:

- Non-typing: in this case no types are used, such as the *2000 US Census*.
- Small-medium typing: datasets in which types are used around one of every four subjects ($\approx$ 25%). In our study, we find two cases, *Flickr* (31%) and Dbpedia (16%), matching this scenario.

Dataset	# Classes	# Typed Subjects	Ratio
		($\|S_G^C\|$)	($\frac{\|S_G^C\|}{\|S_G\|}$)
Jamendo	11	290,291	86.42%
LinkedMDB	53	665,441	95.83%
Dbtune	64	10,042,747	80.96%
Flickr	3	1,690,338	30.79%
SWDF	62	8,916	85.11%
Faceted DBLP	14	3,591,091	100.00%
Wordnet	25	873,986	79.42 %
Dbpedia 3-8	351	4,007,892	16.17%
2011 Australian Census	15	51,768	100.00%
2000 US Census	0	0	0.00%
AEMET	5	394,289	100.00%
Ike	12	114,471,666	99.99%
Linked Geo Data	1081	49,352,200	95.06%
Affymetrix	3	1,421,291	99.97%

Table 4.6: Number of classes, typed subjects and its ratio for the experimental framework.

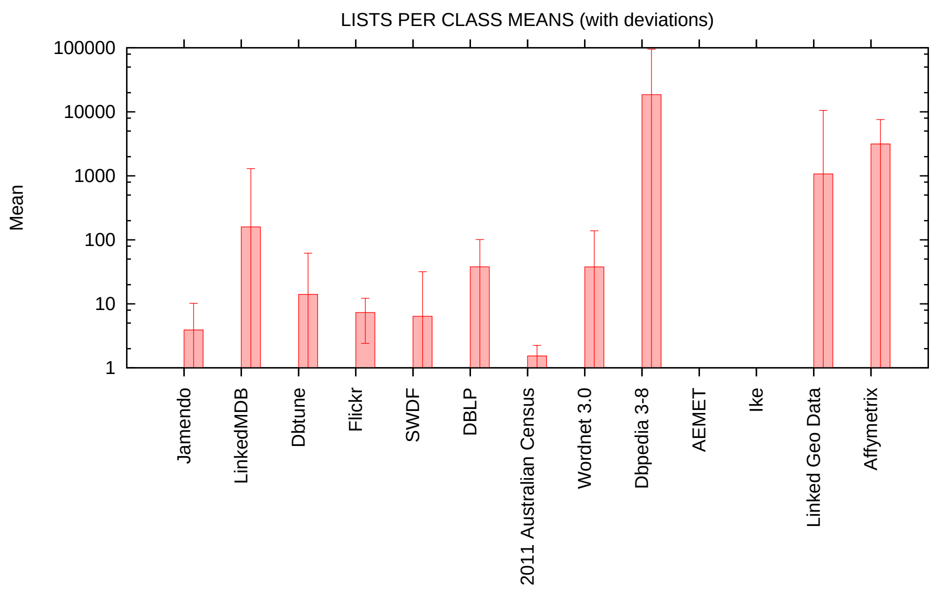


Figure 4.23: Mean lists per class for the evaluation datasets. The y-axis is in logarithmic scale.

- Extensive-typing: most subjects are typed. This is the case of most datasets in our study, ranging from 79% to 100% of typed subjects.

Next, we extend our previous study on predicates, performing a mean of predicate lists per class (see Definition 25). This is represented in Figure 4.23 (in logarithmic scale). Note that the mean is exactly 1 (with no deviation) for *AEMET* and *Ike*.

The mean figures show that, seven of thirteen datasets hold a mean of less than ten predicate lists per class, and it remains valid independently of the size of the dataset. This means that, given a class, we can automatically state that all subjects of this class are described with one of ten variation of predicates, on average. Another three datasets range between 10 and 100 lists per class (which remains still small). The three datasets with more different lists, obviously present more lists per class (up to 19,000 for *Dbpedia*). Nevertheless in these latter cases the deviation is also high, hence we can also find classes with much lesser variants.

Finally, we present in Figure 4.24 a brief comparison of mean out-degrees for typed subjects with respect to all subjects. We restrict to those datasets having small-medium typing (as defined above),

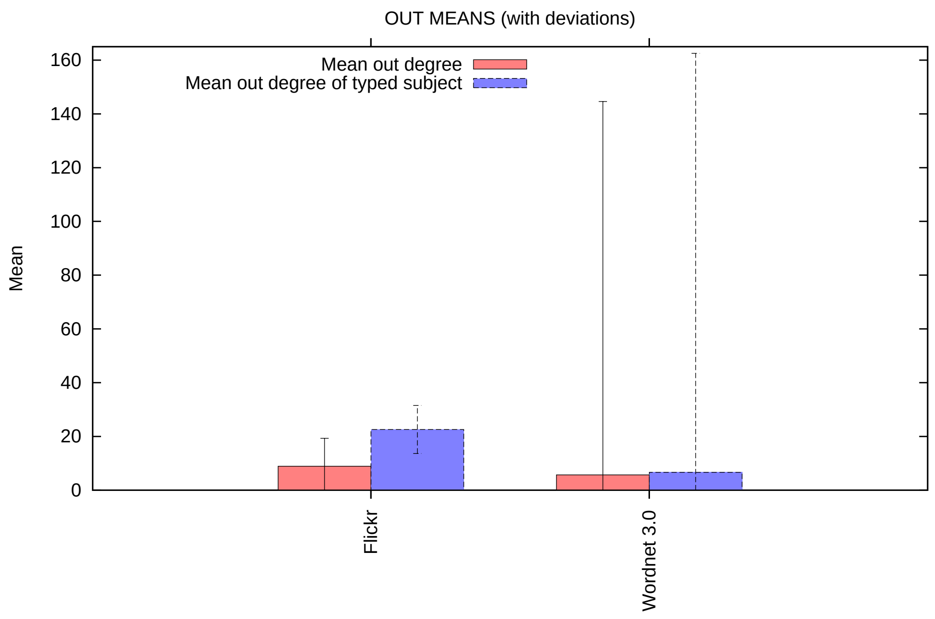


Figure 4.24: Mean out degree for the evaluation datasets in comparison with typed subjects.

as in the extensive-typing case both means are similar and the comparison makes no sense. Thus, we compare the figures of *Flickr* and *Wordnet*. We extend the range of the y-axis to show that both means and deviations are comparable. Nevertheless, typed subject are, in fact, described with slightly more triples on average than all subjects without restrictions. This can be seen as a way of providing a detailed description for such important nodes to navigate and organize the information of the graph.

I'm sorry but I don't want to be an Emperor - that's not my business - I don't want to rule or conquer anyone. I should like to help everyone if possible.

The Great Dictator (1941)

5 Discussion

This chapter briefly summarizes the contributions (§5.1) of this part of the thesis devoted to characterize the RDF structure. We also list the most important empirical findings(§5.2) and we envision potential applications (§5.2).

5.1 Contributions

In this part of the thesis we have studied and characterized the real structure of RDF datasets. First, in Chapter 3, we have motivated our purpose in the sparingly number of previous empirical studies and the few parameters considered. We have reviewed the state-of-the-art baseline revealing power law distributions, and the existence of a small-world phenomenon.

Next, in Chapter 4, we propose and define novel metrics for RDF aimed at characterizing real-world RDF data. Our initial purpose was to provide a toolkit of parameters that could both i) help determine common features in most RDF datasets when possible and ii) become a useful handbook when developing or optimizing RDF data structures (such as the ones proposed in the next parts of this thesis), indexes and other related technologies.

The proposed metrics cover a wide spectrum of parameters. First, the RDF dataset is regarded as a graph labeled with predicates, and we give metrics to characterize the subject (out-) and object (in-) distributions. We measure their degree (*out- and in-degrees* respectively), the presence of multivalued pairs (*partial degree*), the number of different predicates per node (*labeled degree*) and the direct relationships disregarding labels (*direct degree*).

Then, we characterize the distribution of predicates, which is of great importance as they hold the semantics of the datasets. We define their cardinality (*predicate degree*), and the distribution of subjects and objects per predicate (*predicate in and out-degre*). This later is equivalent to describe the domain and range of each predicate.

We consider the repetitions of nodes playing different roles, hence common ratios are defined: *subject-object*, *subject-predicate* and *predicate-object*. Given the importance of the first ones as hubs in the navigation of the graph, we propose to characterize the subject and object degrees restricted to common subject-objects.

Agreeing that the list of predicates per subject can be repeated in several subjects, we then focus on parameterize these list and their repetitions. We define a *ratio of repeated predicate lists*, the cardinality of each list (*predicate list degree*) and the number of lists in which each predicate takes part (*lists per predicate degree*).

Finally, we make a special distinction of typed subjects, as they could share commonalities. We count the number of classes, typed subjects and their ratio over the total number of subjects. We also define the number of different predicate lists per class (*lists per class degree*) and we propose to consider the subject and predicate list degrees restricted to typed subjects.

5.2 Result Summary

As we have motivated, the generalization of common patterns, when possible, was one of the intended purposes of the proposed metrics. Nonetheless, evaluating this generalization is not an easy task given the huge range of different types of RDF datasets, changing in size, domain, authoring and conversion tools, and modeling. Thus, when no generalization is possible, the focus is not to serve as a one-size-fits-all set of metrics, but to provide a simple set of useful metrics for a given scenario.

From these initial premises, we established an evaluation framework consisting in fourteen datasets trying to cover a wide range of different datasets. The following summary of conclusions can be drawn from the evaluation results:

- As we expected, subject-predicate and predicate-object ratios are almost negligible and subject-object is the most frequent path constructor. Datasets were grouped in three ranges, near 0% of shared entities (4 dataset), [20-30]% (5 datasets), 60% (3 datasets) and near 100% (2 datasets). The design and domain of a dataset have a strong influence in the presence of such intermediate nodes.

- Most datasets present a mean of 10 triples per subject at most, but with high deviation. In turn, almost all datasets have a lower mean in-degree. That is, given an object, it is present in a mean of 10 triples (also with high deviation).

- All datasets reveal a remarkably skewed structure on objects. Twelve of the fourteen datasets are very clear power law distributions.

- The distribution of subjects is also skewed except for some cases. We identify that the distribution is less skewed when the data modeling follows well-structured patterns (such as census data), in which resources are described with a similar number of triples.

- Most datasets show that each subject is described with less than 10 different predicates, on average. This remains true if we restrict to common subject-objects.

- The number of predicates related to a given object is very close to 1, also for common subject-objects.

- The mean partial out-degree is slightly bigger than 1, which implies that the presence of multivalued pairs $(subject, predicate)$ is not so frequent.

- Although the mean in-degree also remains close to 1, the high deviation denotes pronounced skewed distribution of multivalued $(predicate, object)$ pairs.

- The out- and in-degrees are comparable to their corresponding direct out- and in-degrees. This means that given a subject and an object, if they are related, only one predicate brings these nodes together.

- The results for predicate degrees state that, on average, given a predicate at random, it is probably related with more subjects than objects.

- The number of different predicate lists is spectacularly low and predicate lists are massively repeated in all cases. Over 947‰ of the predicate lists are repetitions. This remains also true for typed subjects.

- The distribution of the repetitions of predicate lists is also skewed. The only exception found in the experimentation was the case in which mostly all predicates are present in every entity, thus resulting in a non skewed distribution. This latter was only present in 2 of 14 datasets.

- Each predicate participates in a number of predicate lists proportionally small with respect to the number of lists.
- The number of classes remain proportionally small with respect to the number of triples and entities in the dataset.
- Most datasets are extensively typed, that is, [80-100]% of the subjects are typed.
- Half of the datasets holds a mean of less than 10 predicate lists per class, and it remains valid independently of the size of the dataset.

5.3 Applications

We expect that these metrics and observations can provide insights to take advantage of some of the revealed features. In future parts of this thesis, we exploit the skewed structure of RDF graphs. We will motivate our decisions in most of these metrics, specially those delimiting the mean degrees (total, labeled, partial, etc). In particular, the binary deployment proposed in Section 7.2 takes advantage of the subject-object ratio characterization and groups the references to the same node. In turn, we will represent the graph compacting the distribution with implicit and coordinated adjacency lists, parametrized by the degree metrics.

We also expect to help develop and optimize better dataset designs, visualizations, efficient RDF data structures, indexes (in particular, structural indexes) and compression techniques. The full list of optimizations and decisions are subject of each particular scenario and are out of the scope of this study. Nonetheless, through the previous chapter, we have introduced some concrete decisions which can be considered.

For instance, as the number of predicates per object is 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 can obtain important results.

In turn, the number of few predicates per subject and their distribution (*labeled out degree*) is a clear indicator of the presence of star-shaped nodes. Together with the characterization of intermediate nodes, this could serve query suggestion and visualization purposes. In particular, it is highly remarkable that intermediate nodes are reached by a mean of one solely different predicate, reducing the number of predicates which connects different parts of the graph.

The family of indexing techniques following vertical partitioning can consider also the predicate distributions and, potentially, make use of these metrics to optimize the resolution of complex queries.

Predicate lists and the characterization of classes would serve several purposes such as visualization, structural indexing for querying and reasoning. We would like to remark the massive repetition of predicate lists, in general, and the low number of predicates per class in particular. This may help in determining structural indexes for such purposes.

Part II

Binary RDF Representation for Publication and Exchange

- I remember you being a lot bigger. - To a ten year old I'm huge.

Hook (1991)

6 Introduction

The last decade has witnessed an unexpected change in the policies for exposing Open Data thanks to the efforts (and advantages) of the Linked Open Data movement (§2.2). As we stated, the *data deluge* and the phenomenon of *Big Data* has no doubt affected the semantic data, actually becoming "Big Semantic Data" (see Definition 6).

This chapter introduces scalability drawbacks managing Big Semantic Data exposed in the Web of Data. First, we describe the different actors within the current Web of Data (§6.1). Then, we define a common workflow arising in this scenario, typically conformed of three correlated processes; publication, exchange and consumption (query) of the information (§6.2). Next, we review the state of the art techniques to perform each process (§6.2.1). Finally, their problems will motivate the need of binary RDF representations (§6.3).

6.1 Stakeholders in Big Semantic Data Management

We have shown that the "data deluge" had been originally devised in the field of *eScience* (§2.3). Although we identify data scientists as one of the main actors in the management of Big Semantic Data, we have rapidly moved to a globalized scenario. We unveil "traditional" users moving from a Web of documents to a Web of Data, or, in this context, to Big Semantic Data exposed in the Web of Data.

Note that the scalability problems arising for data experts and general users cannot be the same, as these are supposed to manage the information under different perspectives, as follows:

- **Effort:** A data expert can make strong efforts to create novel semantic data or to analyze huge volumes of data created by third parties. In contrast, general users expect a treatable information, a knowledge ready for consumption.

- **Response Times:** General users expect to deal with semantic information in reasonable times whereas data experts deal with long batch processing. For instance, a user retrieving all artistic performances located in "Rome" in a given year could expect a response in a range of seconds. In some scenarios, real-time processing is a mandatory requirement. For a data expert, though, it is perfectly accepted to spend several hours performing a closure of a graph.

- **Resources:** Data experts can make use of data-intensive computing, powerful servers or distributed machines and specific algorithms taking advantage of these infrastructures. Users run on generic machines, mobile devices and other configurations with more limited resources.

Although one could establish an isolated categorization of the problems of these worlds, we cannot forget that the **V**alue of Big Semantic Data exposed in the Web of Data is establishing and discovering links between diverse data. This interlinkage is beneficial for all parties. For instance, in life sciences it is important to have links between the bibliographic data of publications and the concrete genes studied in each publication, thus another researchers can look up previous findings of the genes they are currently studying (Hey et al., 2009). Thus, the "v"ariety of both "publications" and "genes" is represented, linked and managed under a semantic perspective.

Our concern here is clear: *To address user-specific management problems while remaining in the general open representation and publication infrastructure of the Web of Data*.

This premise leverages the Web of Data to exploit the full potential of Big Semantic Data. In turn, user management problems have to be specifically addressed, hence they need a prior analysis. This section provides an approach toward this characterization. We first establish a simple set of stakeholders in Big Semantic Data management, from where we define a common data workflow in order to better understand the main processes performed in the Web of Data. In other words, this first characterization of the involved users and processes would allow researchers and practitioners to clearly focus their efforts on a particular area.

6.1.1 Participants and Witnesses

The Web of Data has successfully emerged on the roots of Open Data. However, the initial main cornerstone which feeds all the infrastructure (semantic data creation) is one of the hardest task for a common user. To date, neither the creation of self-described semantic content nor the linkage to other sources, are simple tasks for a common user. There exists several initiatives to bring semantic data creation to a wider audience, being the most feasible the use of RDFa (Adida, Herman, Sporny, & Birbeck, 2012), a way to include RDF data within HTML pages.

Vocabulary and link discovery can also be mitigated through searching and recommendation tools (Hogan, 2011; Volz, Bizer, Gaedke, & Kobilarov, 2009). However, in general terms, one could argue that the creation of semantic data is still almost as narrow as the original content creation in Web 1.0. In the LOD statistics, previously reported, only 0.42% of the total data is user-generated. It means that public organizations (governments, universities, digital libraries, etc.), researchers and innovative enterprises are the main creators, whereas citizens are, at this point, just witnesses of a hidden reality.

All this brings up two main facts. On the one hand, the Web of Data success, beyond the data expert community, strongly relies on achieving the general audience to implicate in such creation of machine readable descriptions (RDF). On the other hand, the current reality shows that few creators have been able to produce huge volumes of RDF data and to feed the system. One could argue, though, about the quality of these publication schemes (in agreement with empirical surveys (Hogan et al., 2012)).

In what follows, we characterize a minimum set of stakeholders interacting with Big Semantic Data in the Web of Data. Figure 6.1 illustrates the main identified stakeholders. We provide a classification at two orthogonal levels, according to the stakeholder role and nature.

On the first level, three main roles are identified: *creators*, *publishers* and *consumers*, with an internal subdivision by the creation method or intended use. In parallel, we distinguish between *automatic stakeholders*, *supervised processes*, and *human users*. We describe below each stakeholder, acknowledging that this classification may not be complete as it is intended to cover the minimum foundations to understand the processes in Big Semantic Data. In turn, categories are not disjoint. For instance, a creator can also consume information and vice versa.

Creator. As stated, the creator feed the Web of Data with new content. We define the process of creation as the generation of a distinguishable new RDF dataset by, at least, one of these processes:

- *Creation from scratch:* the new dataset is not based on a previous data model. Even if the data exist beforehand, the data modeling process is not influenced by the previous data schema. RDF authoring tools[1] are traditionally used.

[1]A list of RDF authoring tools can be found at *http://www.w3.org/wiki/AuthoringToolsForRDF.*

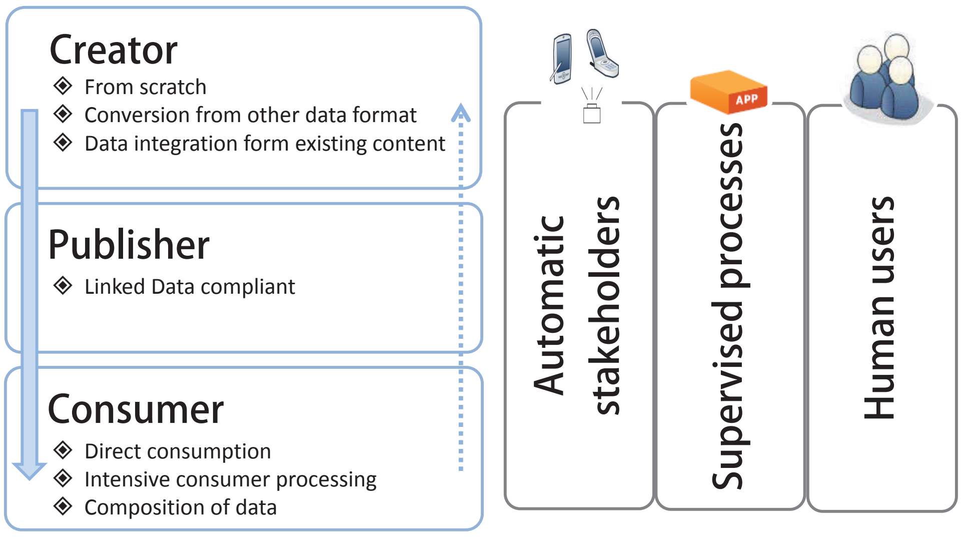


Figure 6.1: Stakeholder classification in the Web of Data.

- *Conversion from other data schema*: the creation phase is highly determined by the conversion of the original data source; potential mappings between source and target data could be used; *e.g.* from relational databases (Arenas, Bertails, Prud'hommeaux, & Sequeda, 2012), as well as (semi-) automatic conversion tools[2].
- *Data integration from existing semantic content*: the challenge is to achieve an efficient integration of vocabularies and the validation of shared entities (Knoblock et al., 2012).

As stated, the creator can construct a new dataset combining these three tasks. For instance, a new RDF dataset describing a city can be created by means of a sort of: i) creation from scratch for those facilities never modeled before (*e.g.* cultural events), ii) conversion of some existing data (*e.g.* transport) and integration of other semantic data (*e.g.* weather data).

Note that several subtasks can also be shared among all three processes. In particular, two main tasks are the identification of those entities to be modeled and the reuse of vocabularies. The first one is even more important in the creation from scratch, as no prior identification has been done. The latter is crucial in data integration in which different ontologies could be aligned.

A complete description of the creation process is out of the scope of this thesis. A detailed guide for Linked Data creation can be found in Heath and Bizer (2011).

Publisher. A publisher is one that makes RDF data publicly available for different purposes and users. In the context of the Web of Data, let us suppose that the publisher follows all the Linked Data principles (§2.2). We distinguish between creators and publishers as the roles can strongly differ in several scenarios. The idea is that publishers hold RDF content, possibly created by third-parties. Publishers, then, are responsible of the publication scheme and policy, and the availability of the offered services (such as querying). For instance, a creator could be an automatic system of sensors reporting temperature measures in RDF (Atemezing et al., 2012), while the publisher is the agency exposing this information in the Web of Data. It is worth noting that the publisher provides entry points to the information, dealing with correct HTTP URIs and their dereferenciation, in compliance with the principles of Linked Data.

Consumer. In general terms, a consumer makes use of published RDF data for disparate purposes. As for the traditional Web, the computational task required for consumption can be distributed between

[2]A list of RDF converters can be found at *http://www.w3.org/wiki/ConverterToRdf*.

the server (the publisher) and the client (final consumer). According to the distribution, we distinguish between two main types of consumptions:

- *Direct consumption*: a process whose computation task mainly involves the publisher, without intensive processing at the consumer. Downloads of the total dataset (or subparts) and online tasks of querying, information retrieval, visualization or summarization, are simple examples in which the computation is focused on the publisher.

- *Intensive consumer processing*: processes with a non-negligible consumer computation, such as offline analysis, data mining or reasoning over the full dataset or a subpart of it (*e.g.* live views (Tummarello et al., 2010)).

Together with this characterization, a special type of consumption is the *composition of data*. That is, we refer to processes consuming different data sources and services in order to serve their purposes. RDF snippets in search engines (Haas, Mika, Tarjan, & Blanco, 2011) and federated services on top of existing publishers in the Web of Data (Schwarte, Haase, Hose, Schenkel, & Schmidt, 2011; Taheriyan, Knoblock, Szekely, & Ambite, 2012) are two examples of these consumers.

As shown in Figure 6.1, the second level of classification of the stakeholders regards the nature of creators, publishers and consumers. Three main types of stakeholders are identified: *automatic stakeholders*, *supervised processes* and *human stakeholders*.

- **Automatic stakeholders**, such as sensors, Web processes (crawlers, search engines, recommender systems), RFID labels, smartphones, etc. Automatic RDF streaming, for instance, would become a hot topic, specially within the development of smart cities (De et al., 2012).

- **Supervised processes**, such as semantic tagging and folksonomies within the domain of social networks (García-Silva, Corcho, Alani, & Gómez-Pérez, 2012). These are automatic processes requiring some sort of human supervision.

- **Human stakeholders,** who currently perform most of the task for creating, publishing or consuming RDF data.

Example. The following running example provides a practical review of this classification. Nowadays, an RFID tag can document a user context through RDF metadata descriptions (Foulonneau, 2011). Let us imagine a system in which sensors provide georeferenced information about pollution in different parts of a city. We could have thousands of sensors providing RDF excerpts. In turn, citizens can visualize and query online this information which has been linked to other data (*e.g.* weather) or facilities and industries of the city. For instance, one could establish potential correlations between the pollution levels of a given area and the environmental plan of the city council or other unexpected events such as strokes, massive live concerts or sport matches. In addition, RDF data can be consumed by a monitoring system to automatically alert population in case of extreme pollution levels in a particular area. When this system is integrated with census data, the possibilities are even higher.

Following the classification, sensors are *automatic creators* conforming, all together, a potentially big semantic dataset. A sensor should be designed to take care of RDF descriptions, *i.e.*, to follow a set of vocabularies and description rules and to minimize the size of descriptions. Additionally, automatic intermediate hubs would collect data of several sensors. In any case, it is clear that sensors can not address all publishing policies such as providing query endpoints and other services to users. The reasonable configuration is that the authoritative organization in charge of the system will be responsible of its publication, applications and services over these data. This publication authority would implement a *supervised process* collecting the information, filtering it (*e.g.* eliminating redundancy) and finally publishing in compliance with Linked Data standards. This process should be carefully designed and implemented to solve scalability issues of huge RDF datastreams. Although it could be automatic, let us suppose that human intervention is needed, for instance to link sensor data to information about city events. Note also that intermediate hubs could be seen as *supervised consumers* of the sensor data, yet the information coming from the sensors is not openly published but streamed to the appropriate hub. The final target are *human consumers*, in case of the online users (concerned of query resolution, visualization, summarization, etc.) or an *automatic consumer*, in case of monitoring (doing potential complex inference or reasoning).

This feasible example agrees with our initial premise of the enormous diversity of involved actors and their different concerns. When designing a system, this classification could help as a first step in the identification of the roles and natures of the stakeholders. Then, different scalability issues should be considered for each kind of stakeholder.

6.2 The Workflow of Publication-Exchange-Consumption

We henceforth consider the creation step out of the scope of this work, because our approach relies on the preexistence of big semantic data (without belittling those ones which can be created hereinafter). Although very interesting issues arise in this phase, we focus on tasks involving large-scale management as they take part in most scenarios. For instance, scalability issues of visual authoring a big RDF dataset are comparable to RDF visualization by consumers, or the performance of RDF data integration from existing content depends on efficient access to the data and thus existing indexes, a crucial issue also for query response.

Management processes for publishers and consumers are diverse and complex to generalize. However, it is worth characterizing a common workflow present in almost every application in the Web of Data in order to place scalability issues in context. Figure 6.2 illustrates the identified workflow of Publication-Exchange-Consumption.

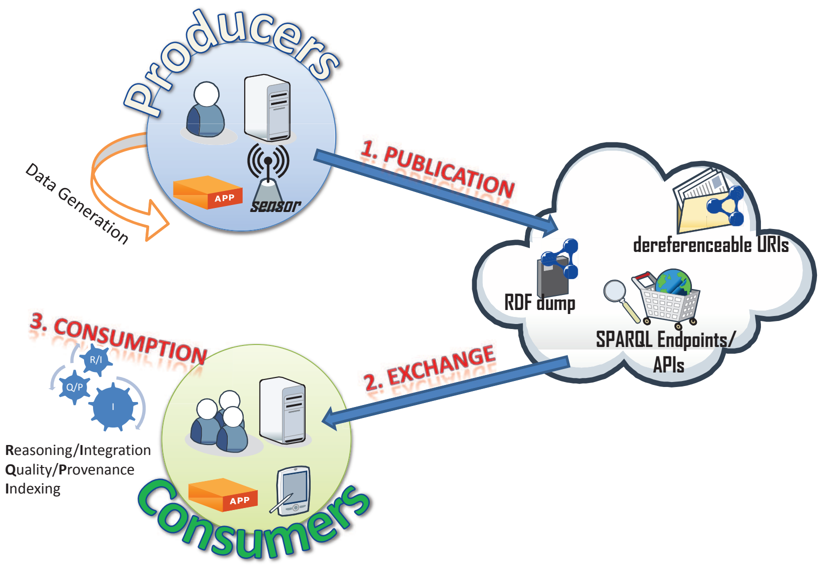


Figure 6.2: Publication-Exchange-Consumption workflow in the Web of Data.

- **Publication** refers to the process of making RDF data publicly available following the Linked Data principles. Strictly speaking, the only obligatory "service" in accordance with the principles is to provide dereferenceable URIs, *i.e.*, related information of an entity. In practice, publishers used to complete this limited functionality. At the most basic level, they provide RDF dumps in order to download the complete RDF dataset, or at least some parts of it. A recommended practice is to go one step further and expose data through public query APIs. Typically, queries are written in the SPARQL query language and posed via SPARQL endpoints, which interprets SPARQL queries and serves its results.

- **Exchange** is the process of information interchange between publishers and consumers. Although the information is represented in RDF, note that consumers could obtain different "views" and hence formats, some of them not necessarily in RDF. For instance, the result of a SPARQL query could be provided in a CSV file or the consumer would request a summary with statistics of the dataset in a XML file. As we are issuing management of semantic datasets, we restrict exchange to RDF interchange. Thus we rephrase exchange as the process of RDF exchange between publishers and consumers after an RDF dump request, a SPARQL query resolution or another request or service provided by the publisher.

- **Consumption** can involve, as stated, a wide range of processes, from direct consumption to intensive processing and composition of data sources. Let us simply define the consumption as the use of potentially large RDF data for diverse purposes.

A final remark must be done. As we stated when defining Big Semantic Data (§2.3), we do not restrict management to large RDF datasets. We open scalability issues to a wider range of publishers and consumers with more limited resources. For instance, similar scalability problems arise when managing RDF in mobile devices; although the amount of information could be potentially smaller, these devices have more restrictive requirements for transmission costs/latency, and for post-processing due to their inherent memory and CPU constraints (Le-Phuoc et al., 2010). Thus, although we provide approaches for managing large RDF datasets, we assume similar decisions could be taken for limited configurations with equivalent scalability issues.

6.2.1 State of the Art

This section summarizes some of the current trends to address publication, exchange and consumption at large scale.

Publication schemes. Current straightforward publication of Big Semantic Data presents several problems, at all levels:

- *RDF dumps.* A massive empirical study of Linked Open Data datasets by Hogan et al. (2012) draws discouraging conclusions: few providers attach metadata to their resources (authoring, summary of content, statistics, etc.) or licensing information. In accordance also with the work by Fernández, Martínez-Prieto, and Gutiérrez (2010), the paradoxical fact is that the lack of systematic metadata is so worrying that RDF dumps do not encourage its consumption. Potential users know almost nothing about the content they are going to download beforehand. Thus, managing millions (and billions) of RDF triples in Big Semantic Data is, first, a matter of blind trust.

- *SPARQL endpoints.* Same features can be applied to SPARQL endpoints in which consumers, most times, do not even know which are the vocabularies used in the data modeling. In these cases, to query a dataset is an exploration task in which queries are constructed by "trial and error". In addition, SPARQL endpoints are services built on top of an RDF engine which has to address efficient querying of such big data.

- *Dereferenciation of HTTP URIs.* Deferenceable URIs can be done in a straightforward way, publishing one document per URI, or set of URIs. However, the publisher commonly materializes the output by querying its RDF engine at URI resolution time. This moves the problem again to the underneath RDF store, which is potentially solving other SPARQL queries. The empirical study by Hogan et al. (2012) also confirmed that publishers often do not provide locally-known inlinks in the dereferenced response which must be taken into account by consumers.

In general terms, except for the general Linked Data recommendations (Heath & Bizer, 2011), few works address the publication of RDF at large scale. The Vocabulary of Interlinked Datasets, VoiD (Alexander, Cyganiak, Zhao, & Hausenblas, 2009), is the nearest approximation to the discovery problem, providing a bridge between publishers and consumers. Publishers make use of a specific vocabulary to add metadata to their datasets, *e.g.* to point to the associated SPARQL endpoint and RDF dump, to describe the total number of triples and to connect to linked datasets. Thus, consumers can look up this metadata to discover datasets or to reduce the set of interesting datasets in federated queries over the Web of Data (Akar, Halaç, Ekinci, & Dikenelli, 2012). Finally, the proposal of Semantic Sitemaps (Cyganiak, Stenzhorn, Delbru, Decker, & Tummarello, 2008) extends the traditional Sitemap Protocol for describing RDF data. They include new XML tags so that crawling tools (such as Sindice[3]) can discover and consume the datasets.

RDF Serialization Formats. As we previously stated, we focus on exchanging large-scale RDF data (or smaller volumes in limited resources stakeholders). Under this consideration, the RDF serialization format directly determines the transmission costs and latency for consumption. Unfortunately, datasets are currently serialized in plain and verbose formats such as RDF/XML (Beckett, 2004) or Notation3: N3 (Berners-Lee, 1998).

RDF/XML (Beckett, 2004) was released hand in hand with the latest W3C RDF Recommendation. In fact, it was a good solution to take advantage of all solutions managing XML at that time. However, it terribly overloads the representation with verbose information "for humans", whereas humans should not be the focus when downloading, for instance, hundreds of millions of triples. Same consideration could be argued when optimizing the representation for limited devices, in which we should prioritized the efficiency. RDF/XML includes, though, some naive compacting features, summarized in the list below (Fernández, Martínez-Prieto, Gutiérrez, Polleres, & Arias, 2013):

- Omitting Blank Nodes (Beckett, 2004, section 2.11): The attribute *rdf:parseType="Resource"* allows to implicitly create blank nodes.
- Omitting Nodes (Beckett, 2004, 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 (Beckett, 2004, 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 (Beckett, 2004, section 2.14): 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.

In turn, Notation3 (N3 (Berners-Lee, 1998)) is a language which was originally intended to be a compact and readable alternative to RDF/XML, optimized for reading by scripts. Thus, it reduces verbosity and represents RDF with a simple plain grammar. 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 instance, repetition of another

[3] `http://sindice.com/`

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

Turtle (Beckett & Berners-Lee, 2011) 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 (Grant & Beckett, 2004) 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 (Alexander, 2008) 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 (Weaver & Williams, 2011) 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.

In order to reduce exchange costs and delays on the network, universal compressors (*e.g.* gzip) are commonly used over these plain formats. In addition, specific interchange oriented representations may be also used. For instance, the Efficient XML Interchange Format: EXI (Schneider & Kamiya, 2011) may be used for representing any valid RDF/XML dataset.

Efficient RDF Consumption. The aforementioned variety of consumer tasks hinders to achieve a one-size-fits-all technique. However, some general concerns can be outlined. In most scenarios, the performance is influenced by i) the serialization format, due to the overall data exchange time, and ii) the RDF indexing/querying structure. In the first case, if a compressed RDF has been exchanged, a previous decompression must be done. In this sense, the serialization format affects the consumption through the transmission cost, but also with the easiness of parsing. Once the consumer has downloaded the dataset, the most likely scenario is indexing it in order to operate with the RDF graph, *e.g.* for intensive operation of inference, integration, etc., but also for the most simple query. Current serialization formats do not provide any means of direct access to the data, *i.e.* they only provide sequential parsing.

Although the indexing at consumption could be performed once, the amount of resources required for it may be prohibitive for many potential consumers (specially for mobile devices comprising a limited computational configuration). In both cases, for publishers and consumers, an RDF store indexing the datasets is the main actor for efficient consumption.

RDF is a logical data model which does not limit its physical storage or indexing. However, these proceedings are strongly related with the later querying process, which is typically performed by SPARQL queries (see §2.1.2). The semantics and complexity of the SPARQL query language have been fairly studied theoretically, showing that full SPARQL evaluation is PSPACE-complete[4] (Perez et al., 2009) due to the OPTIONAL operator alone (Schmidt, Meier, & Lausen, 2010). However, our empirical study of real-world SPARQL queries (Arias, Fernández, Martínez-Prieto, & de la Fuente, 2011) reveals that most SPARQL queries are simple. In fact, over a large DBPedia log, up to 66.41% of the queries just contain one single triple pattern (see Definition 3). In other logs, such as the Semantic Web Dog Food[5], the percentage of these simple queries reaches up to 97.25%.

Several RDF indexes and RDF stores explore efficient SPARQL resolution methods. We review the most important approaches in Section 12.2, showing that the vast majority of them suffers from lack of

[4]A problem is PSPACE-complete if it can be resolved taking polynomial space *w.r.t* the input and every PSPACE problem can be converted to it in polynomial time.

[5]`http://data.semanticweb.org`

scalability in Big Semantic Data. There is still a large interest in querying optimization (Schmidt et al., 2010), whose performance is diminished when the RDF stores manage very large datasets.

6.3 Our Goal

Managing Big Semantic Data yields to optimize each process in the Web of Data workflow. In other words, all steps must be designed to address the three Big Data dimensions, volume, velocity and variety. Whereas we already argue that variety is already addressed with RDF and the Linked Data principles, four brief insights can be gleaned from the study of the stakeholders, processes and current state of the art in the Web of Data:

1. Data serialization has a big impact on the workflow, as traditional RDF serialization formats are designed to be human readable instead of machine processable. They may fit smaller scenarios in which volume or velocity are not an issue but, under the presented premises, they become a bottleneck in the workflow. Moreover, current RDF serializations only enable sequential scan.
2. Besides inadequate overweighted serializations, most publishing schemes obviates metadata and other facilities to upgrade publication and enable discovery for consumption.
3. Even for simple operations, current serialization formats do not provide any means of direct access to the data. Thus, offline RDF consumption typically results in a painful set of costly tasks: exchange, decompression, indexing all plain data and, finally, use. Although the information remains semantically the same, note that each of these processes manages different data representations with different levels of functionality.
4. Diverse stakeholders acts in the Web of Data, with different purposes. However, all them are influenced by plain, non-functional, human-readable formats while managing Big Semantic Data.

Moreover, the aforementioned skewed structure of real-world RDF data (characterized in Chapter 4) gives insights showing that a compact RDF representation should be achieved. The motivation and state of the art call for a binary representation for RDF aim at reducing the high levels of verbosity/redundancy and weak machine-processable capabilities of the datasets. We collect the main requirements for an RDF serialization format of Big Semantic, *i.e.* our hypothesis (§1.2) which will be addressed hereinafter.

- **Efficient conversion from and into another RDF format.** In particular, RDF stores must be able to manage such optimized exchange format both to dump their information and to load new one.
- **Clear publication scheme.** For publishing, the format must rely on a clear scheme, providing a standard way to add provenance and other metadata for discovery and processing by consumers.
- **Efficient space.** It must create compressed representations. Big semantic datasets are shared on the Web of Data, and they may be transferred through the network infrastructure. Space minimization reduces, then, both bandwidth costs and latency. In other words, consumers start processing the information faster, which can be essential for real-time processes.
- **Easy parsing.** As stated, consumers are used to perform a sequential triple-to-triple scanning for any post-processing task. This results in several minutes (or hours) when post-processing Big Semantic Data at the consumer. In addition, most of the aforementioned RDF indexes use variants of B-Trees, which are more inefficient to construct on unsorted elements.
- **Ability to locate pieces of data within the whole dataset.** Nowadays, the most basic lookup requires to full scan the plain triples or to re-index the exchanged RDF data at consumer, which was potentially indexed at publisher. We conceive two desirable requirements for our format:

1. It must be ready to solve, natively, a limited core of SPARQL queries, for instance the basic triple patterns. As shown, triple patterns resolution covers a very significant percentage of the real-world SPARQL queries (Arias et al., 2011).
2. It must provide enough flexibility to build additional indexes to efficiently resolve complex SPARQL queries.

In summary, as we argued, "data must be encoded to be the index". The next chapter presents our proposal addressing the core format which fulfills the requirements for efficient publication and exchange. Then, **Part III** and **IV** will present new structures to enhance the initial core with additional query functionality.

You know what's left after the big storm comes and takes out all the big trees? The little trees. The little guys that held on through out the storm.

Very Bad Things (1998)

7 HDT: A Binary Serialization for RDF

Our approach, **HDT**: *Header-Dictionary-Triples* (Fernández et al., 2013), considers the previous requirements, addressing a machine-processable RDF serialization format. It enables Big Semantic Data to be efficiently managed within the common workflows of the Web of Data.

This chapter formalizes the HDT serialization for publication and exchange over a network. First, we present a conceptual description of the HDT logical components: Header, Dictionary and Triples. As HDT allows different implementations for each component, we characterize the requirements, operations and the intended use of each component. Next, we provide a practical deployment of HDT with simple implementations of each component. Then, we design a general RDF/HDT syntax specification and provide specific details for the previous deployment. Finally, we perform an empirical study which analyzes HDT features on real-world datasets.

In the following chapters we present succinct data structures to browse HDT-encoded datasets (direct access to any piece of data) in harmony with all aforementioned requirements.

7.1 Conceptual Description

HDT is designed as an RDF binary encoding which succinctly represents the information of an RDF dataset by organizing and representing the RDF graph in terms of three logical components: Header, Dictionary and Triples (Figure 7.1).

- **Header.** The Header holds metadata describing a big semantic dataset encoded in HDT. Although the binary representation should be machine-oriented, this component is aimed to gather a human-friendly context of the dataset. In spite of the existence of dedicated RDF vocabularies to describe datasets (*e.g.* VoiD (Alexander, Cyganiak, Hausenblas, & Zhao, 2011) and annotation properties in OWL (Motik, Patel-Schneider, & Parsia, 2009, Section 10)), current serialization formats do not provide means on how to publish these metadata along with datasets. In other words, the metadata, when present, is currently provided in the same RDF graph in a nonstandard way which makes difficult to extract and process them automatically.

  In contrast, we propose the metadata to be encoded together with the data but in a distinguishable component, the Header, making metadata a first-class citizen (Fernández et al., 2013). The Header is, by itself, a plain RDF graph, thus leveraging the current semantic infrastructure for management and discovery. The Header triples use standard vocabularies to describe the dataset. For instance, one publisher can provide information about the provenance (authoring, publication dates, version), statistics (size, quality, vocabularies), physical organization (subparts, location of files) and other type of information (intellectual property, signatures).

- **Dictionary.** The Dictionary component organizes the catalog of all different terms used in the dataset (URIs, literals and blank nodes). A unique identifier (ID) is assigned to each term, enabling triples to be represented as tuples of three IDs which, respectively, reference the corresponding terms in the dictionary.

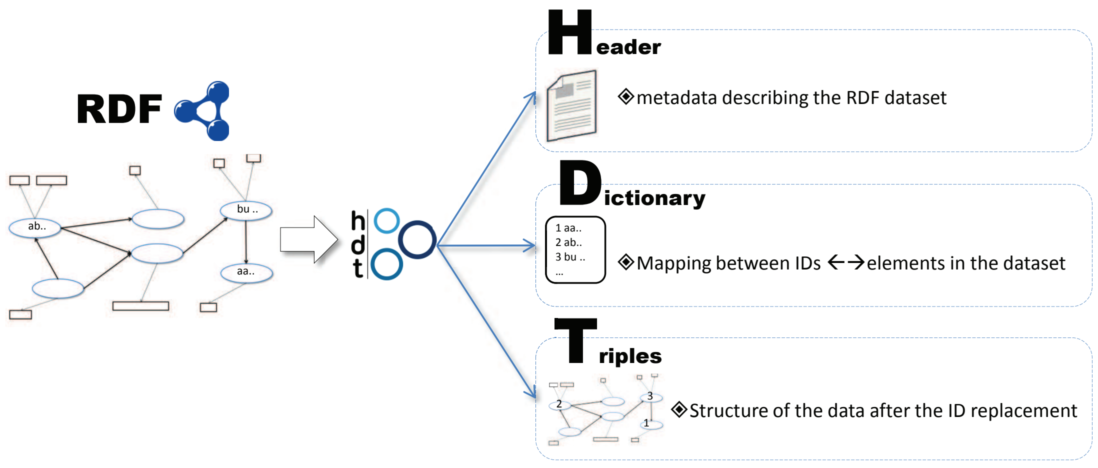

RDF
ab..
bu ..
aa..
hdt
Header
◈metadata describing the RDF dataset
Dictionary
1 aa..
2 ab..
3 bu ..
...
◈Mapping between IDs ←→elements in the dataset
Triples
2
3
1
◈Structure of the data after the ID replacement

Figure 7.1: Description of HDT Components: Header-Dictionary-Triples.

Note that most RDF stores (such as the well-known RDF-3X (Neumann & Weikum, 2010) or Virtuoso (Erling & Mikhailov, 2007)) make use of a dictionary as it allows the graph structure to be indexed as an integer-stream. We propose to incorporate into the binary RDF representation this simple but effective decision for managing RDF. As we will argue, this is a first step toward compactness, since it avoids long terms to be repeatedly represented.

- **Triples.** The triples take advantage of the dictionary mapping to represent a graph of IDs, avoiding to manage nodes and edges with long strings. This is, in fact, the key component to query the RDF structure. On the one hand, an efficient triples encoding can help in triple scanning for post-processing tasks. On the other hand, if data can be easily indexed, basic queries (such as SPARQL triple patterns) could be resolved natively. Ideally, the data exchanged would not need of decompression nor re-indexing to be consumed (or these processes may be performed but in a marginal time *w.r.t* traditional approaches).

Figure 7.2 shows a typical Publication-Exchange-Consumption scenario in HDT. We make use of the following definitions (revised from our previous work (Fernández et al., 2011)).

Definition 26 (HDT processor) *An HDT processor is a component used by application programs to encode their data into HDT and/or to decode HDT data to make the data accessible.*

Definition 27 (HDT encoder) *An HDT encoder is an HDT processor which, at least, is able to encode application data into HDT data.*

Definition 28 (HDT decoder) *An HDT decoder is an HDT processor which, at least, is able to decode and post-process HDT data for the purposes of an application program.*

Definition 29 (HDT core data) *The HDT core data of an HDT representation consists of its Dictionary and Triples components, 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 full dataset.*

The HDT *processor* concept generalizes the notion of publishers and consumers. This is specially useful for environments in which stakeholders can act with several roles, such as consumers which look up and integrate diverse sources and publish big semantic data. In such scenarios, one could distinguish if the involved sources can (or not) manage HDT data, that is, if they incorporate an HDT processor.

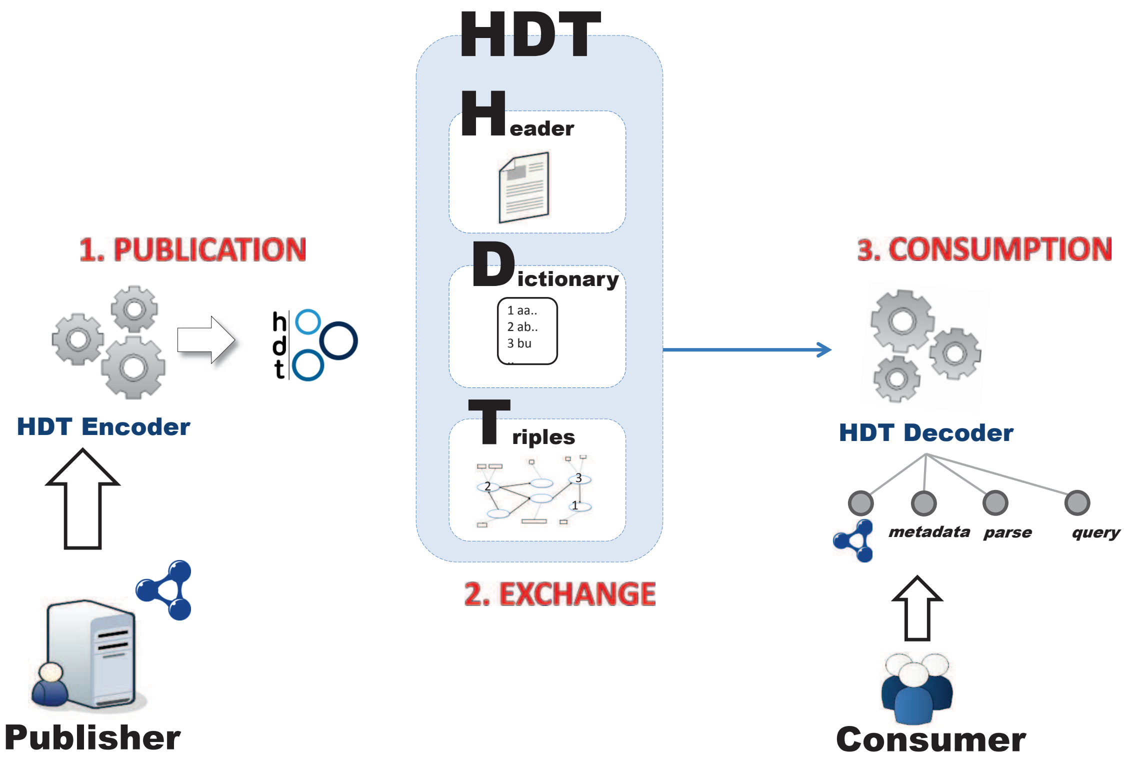


Figure 7.2: The common process of HDT encoding/decoding.

A "pure" HDT consumer (not acting as publisher) may be only interested in decoding and post-processing the exchanged HDT data, including then an HDT *decoder* functionality. In turn, a publisher willing to encode data to HDT, will run an HDT *encoder* software. In both cases they mainly exploit the HDT *core data*, that is, the Dictionary and Triples components, without denying the importance of the metadata of the Header.

Thus, Figure 7.2 illustrates a content publisher making use of an HDT encoder (a program module or an external library) in order to generate HDT from its RDF content. Once published and exchanged, the consumer uses an HDT decoder to efficiently access the HDT header and the core data (dictionary and triples). The HDT decoder should provide the consumer with distinct access possibilities, such as getting the original full RDF dataset, retrieving the metadata of the header, parsing its information into another data structures and querying the data.

Figure 7.3 illustrates a variant of the previous encoding. One could effective argue that a consumer would be interested in downloading only the Header with metadata in order to discover and filter the properties of the dataset. In this case, the Header includes links to the HDT core data. Moreover, the Dictionary and Triples components allow diverse configurations and functionality, which can exploit the trade-off between the compression ratio for exchanging and the natively supported operations. Thus, the dictionary and triples could be split in several chunks or streams, one per different configuration. The user can select the appropriate format for the intended purpose.

The previous HDT basic description was flexible enough to allow this possibility. In order to formalize these variations, in the following we detail the three HDT components and list potential uses and levels of functionality.

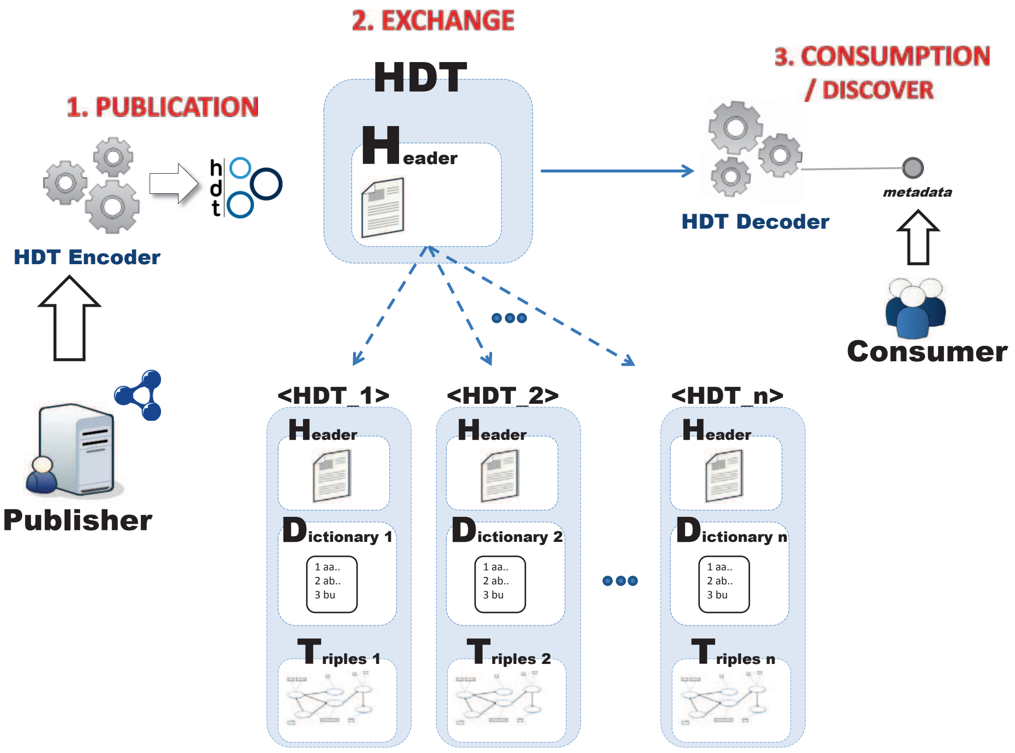


Figure 7.3: A variant of HDT encoding/decoding for discovery.

7.1.1 Header

The Header is an RDF graph describing the dataset. The use of RDF in the Header provides flexibility in the metadata itself. This way, publishers can include the set of properties of their choice to describe the dataset. We distinguish four general types of metadata:

- **Publication Metadata** provides information about the publication act itself, for instance when was the dataset generated, when was it made public, who is the publisher, where is the associated SPARQL endpoint, which is the version of the publication, etc. Many properties of this type can be described using the popular Dublin Core Vocabulary[1].

- **Statistical Metadata** provides statistical information about what follows in the dataset. This class of metadata is valuable for humans to get a glimpse of the content but also to processes such as visualization, indexing optimization for RDF engines or federated query evaluation.

  Metadata can be simple (such as the number of triples, the number of different subjects, predicates, objects, etc.), aggregated (histograms) or slightly richer such as our metrics in Chapter 4.

- **Format Metadata** describes the concrete format of the HDT dataset, *i.e.*, which specific Dictionary and Triples implementations are used. Format metadata also allows i) to state that the publisher provides different available dictionary or triples representations if needed, for example with different space/time tradeoffs, and ii) that the information has been split in several streams. In both cases, format metadata points to the URI locations of each representation.

- **Additional Metadata** collects other informations provided by the publisher using any RDF vocabulary, *e.g.* tags, annotations, or signatures. It also holds specific application metadata.

[1] `http://dublincore.org/`

Header Uses and Operations for Consumption

The Header serves as an entry point for a consumer, who can look up certain properties to have an idea of the contents of the dataset.

Physically, the header can i) precede the `HDT` core data and be downloaded together with the rest of the information, or ii) be a standalone file, downloaded alone. In the first case, it may not serve to decide if a (potentially huge) dataset worth to be downloaded, as the whole dataset has already been downloaded. Nevertheless, the metadata could help discriminate whether a dataset deserves to be really consumed. In addition, features such as statistics could optimize consumption processes such as indexing. In the latter case, the consumer can discover and filter the properties of a given dataset, for instance, through SPARQL queries toward the RDF graph of metadata. This process is done before retrieving the whole dataset. Moreover, if the header metadata can be retrieved and queried online, the user could consume the header online. Last, if the `HDT` core data is distributed in several chunks and available in different formats the user can discriminate the relevant chunk and format to download.

Publishers need simple operations over the Header as it is a general (and typically small) RDF graph. Thus, the set of operations provided by `HDT` encoders can be reduced to the following simple set:

- `write(RDF header, HDT core data)`: Include an RDF description of the header within the `HDT` core data, conforming an `HDT` dataset.

- `update(RDF header, HDT dataset)`: Update an `HDT` dataset with a novel Header. Publishers typically write the Header once, but it could be updated several times with newer information (Fernández et al., 2013).

In turn, consumers can download and access the Header locally, or they might consume it using SPARQL queries. In the first case, `HDT` decoders should provide an operation such as:

- `extract( HDT dataset, RDF header)`: Extract the header out of an `HDT` dataset.

Both the filtering in this case as well as the SPARQL query in the latter case are general operations a semantic library can deal with.

7.1.2 Dictionary

Historically, a dictionary is a repository of information about data such as meaning, relationships to other data, origin, usage, and format (IBM, 1993). Nonetheless, `HDT` makes a simpler conceptualization: the `HDT` dictionary maps each term used in a dataset to a unique integer ID. Thus, it contributes to compactness and more efficient triples management since each term occurrence is now replaced by its corresponding ID, whose encoding requires less bits in the vast majority of the cases.

To the best of our knowledge, the dictionary has not been proposed in any RDF representation syntax. Current RDF syntaxes achieve compactness by means of elementary "dictionaries" for namespaces and prefixes (Fernández et al., 2013). Other approaches exploit the dictionary construction apart from the RDF stores (Martínez-Prieto, Fernández, & Cánovas, 2012a; Urbani, Maassen, & Bal, 2010). A detailed state of the art of RDF dictionaries is addressed in Chapter 9.

Dictionary Characterization for Exchanging

The Dictionary component in `HDT` allows multiple implementations. It is clear, though, that `HDT` encoders and decoders must agree on how to manage a specific encoding. In the following, we distinguish a set of properties that typically characterize a dictionary implementation:

- **Mapping Function.** As we described, the dictionary mapping assigns an ID to each term. Obviously, this assignment is not chosen at random, but it follows a clear pattern. For instance, one could sort all the terms used in a graph by alphabetic order and assign correlative IDs. In turn, an important decision concerns the dictionary partitions. Instead of holding a global mapping, one could distinguish into the different sets of subjects, predicates and objects, because each component in a triple can be then independently named. Thanks to these partitions and due to the limited number of different predicates (studied in Chapter 4), the range of IDs for predicates is limited too. Thus, the ID-stream in triples can make use of fewer bits per predicate. In addition, RDF engines usually map shared subject-object elements with the same ID (Atre et al., 2010).

- **Terms encoding.** Several decisions affect the specif encodings of terms. First, dictionaries usually make use of namespaces and prefixes, already present in most RDF syntaxes. This allows to abbreviate long and repeated strings. Thus, the `HDT` encoder must share this information so that the decoder can undo the abbreviation. Next, a mechanism to separate the serialized terms must be established. Typically, a reserved character delimits terms and dictionary partitions (if present). Finally, the encoding of each term can strongly differs (plain, differential encoding, etc.) and must be defined to enable the correct deserialization.

All these issues must be clearly formalized when designing and exchanging a novel `HDT` dictionary.

Dictionary Uses and Operations for Consumption

For publication and exchanging, the main goal of the Dictionary is to contribute to compactness. Then, once the information is exchanged, the consumer needs two main operations over the mapping (further developed in §9.1):

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

In order to serve these operations, consumers typically load the exchanged dictionary into a functional data structure. This is usually referred to as *parsing*. For instance, consumers could load the serialization into Hashes, B-Trees or other well-known traditional forms of dictionaries. An "intelligent" encoding for the dictionary could help make this parsing more efficient. For instance, a lexicographically order in the encoding could alleviate the posterior sorting made by some structures such as B-Trees.

Besides `locate` and `extract` operations, more advanced techniques might also provide the following operations at consumption time:

- `prefix(p)`: finds all terms starting with the prefix 'p'.
- `suffix(s)`: finds all terms ending with the suffix 's'.
- `substring(s)`: finds all the terms containing the substring 's'.
- `regex(e)`: finds all strings matching the specified regular expression 'e'.

For instance, these advanced operations are very convenient when serving query suggestions to the user, or when evaluating SPARQL queries. As shown in Chapter 2, FILTER operations in SPARQL restrict the final result by a given condition, typically a regular expression but also language or datatype selection (§2.1.2). All these can be evaluated first over the Dictionary which can delimit a range of IDs satisfying the condition (we describe these possibilities in §10.4).

In Part III, we study compressed rich-functional encodings for dictionaries which provide all these operations natively once they are loaded at consumption time.

7.1.3 Triples

The Dictionary mapping allows the RDF graph to be encoded as a graph of IDs. As we will show, the triples organization is the cornerstone to i) exploit inherent graph redundancy and ii) allow triples to be efficiently traversed.

Triples Characterization for Exchanging

Once again, `HDT` devises multiple configuration for Triples encoding, varying in space/time tradeoffs and diverse functionalities. A novel implementation has to clearly define two main properties for a correct serialization/deserialization process:

- **Triples organization.** After ID replacement, the RDF graph is managed as a graph of IDs. Nevertheless, the serialization of such ID-graph can be made in many different ways. For instance, a triples component could perform an in-order traversal, seeing the serialization as a continuous ID-stream of three IDs per triple. In contrast, one could make use of traditional concepts of adjacency lists or other types of structures to achieve compactness.

- **ID-terms encoding.** As we have described in the dictionary, there are different mappings affecting the potential range of IDs. Then, one could codify every ID with the same number of bits (*e.g.* 23 bits) or to leverage the range in each partition to use fewer bits (*e.g.* $log(|P|)$ for predicates). These and other decisions taken to promote compactness, such as using differential and VByte encoding (Williams & Zobel, 1999), must be explicitly known and shared by `HDT` encoders and decoders.

Triples Uses and Operations for Consumption

Similarly to the previous remark in dictionaries, an "intelligent" encoding for the `HDT` triples component can improve parsing at consumption time. We distinguish here four different levels of triples functionality (revised from our previous work (Fernández et al., 2013)) at consumption time:

L0 *Exchange.* At the most basic level, an RDF Triples component solely serves to encode the set of RDF statements, optimizing the objective of exchange. Then, it must allow the minimum operative to retrieve all triples:

- `scanTriples(HDT dataset)`: Returns a sequential scan of all RDF statements in an `HDT` dataset.

L1 *Triple Pattern Search.* An RDF Triples encoding under this level provides basic *triple pattern* resolution[2], serving a search operation such as:

- `searchTriplePattern(triplePattern, HDT dataset)`: Returns the solution for the given triple pattern in the `HDT` dataset.

Ideally, RDF Triples component should be able to resolve efficiently all kinds of triple patterns (see Definition 3). However, this ability can be achieved at the cost of more complex structures. For instance, if a lightweight structure organizes data by subject, it can excel for triple patterns providing a constant subject. In contrast, the performance may be significantly degraded if the subject is not provided in the query. For this reason, we refer to the level "**L1+** *Full Triple Pattern Search*" when the encoding is able to efficiently resolve all triple pattern combination.

[2]In these operations we consider a prior ID replacement of the triple patterns

L2 *BGP Resolution.* In this case, the Triples facilitates to resolve SPARQL BGPs (see Definition 4). As stated, BGPs imply matching two or more triples patterns which share one or more variables, being one of the most common constructions in RDF queries. Thus, RDF Triples component serves an operation:

- `resolveBGP([triplePatterns]?, HDT dataset)`: Returns the solution for the given BGP of one or more triple patterns in the HDT dataset .

L3 *Full SPARQL.* Ideally, the engine should be able to answer efficiently any SPARQL 1.0 query, serving:

- `resolveQuery(SPARQL query, HDT dataset)`: Returns the solution for the given SPARQL query in the HDT dataset.

Note that, compared to the previous level, full SPARQL involves resolving Graph Patterns (see Definition 5). That is, this level must address the *OPTIONAL* and *UNION* operators (shown in Chapter 2.1.2). Thus, as we also refer to "efficient" resolution, the triples component must consider query evaluation optimization techniques.

In Part IV, we study a rich-functional encoding for triples which provides a *L1* level natively once it is loaded at consumption time. Moreover, we propose additional succinct indexes which can be built on top to provide higher levels of functionality.

7.2 Practical HDT Deployment for Publication and Exchange

HDT is designed as a modular format in which different implementations can be plugged into components as long as they provide the minimum basis. In this section we provide a practical HDT deployment aimed at clean publication and compact exchange.

7.2.1 A Specific Vocabulary for the Header

The Header component is always an RDF graph itself in order to take advantage of current applications and services for management and discovery. A practical deployment, though, deals with the appropriate standard vocabularies to describe the dataset. We propose a specific `hdt` vocabulary, with the namespace `http://purl.org/HDT/hdt#hdt`. The mandatory structure of this practical Header is illustrated in Figure 7.4. The explanation of its main features is guided by a running example of a Header in Figure 7.5. This Header is given in Turtle syntax (Beckett & Berners-Lee, 2011) and it corresponds to the RDF graph in Figure 4.1. Note that the choice of a specific RDF syntax for the Header is an issue of the HDT syntax (we address it in §7.3).

First of all, the Header describes an HDT dataset, which is of type `hdt:Dataset` (line `10`). As seen in the structure (Figure 7.4) the `hdt` vocabulary states that:

(hdt:Dataset,rdfs:subClassOf,void:Dataset)

, an HDT description is then an extension of the Vocabulary of Interlinked Datasets, VoiD (Alexander et al., 2009). Thus, the Header can make use of VoiD properties to describe the HDT dataset in a standard way. In addition, it enhances the VoID Vocabulary to provide a standardized binary dataset description.

Next, the Header distinguishes four "sections", corresponding to the four basic types of metadata detailed in Section 7.1.1: *Publication, statistical, format* and *additional Metadata*. In practice, we model these sections by means of four blank nodes (lines `11-14`) grouping the metadata of each type. These are the most important remarks.

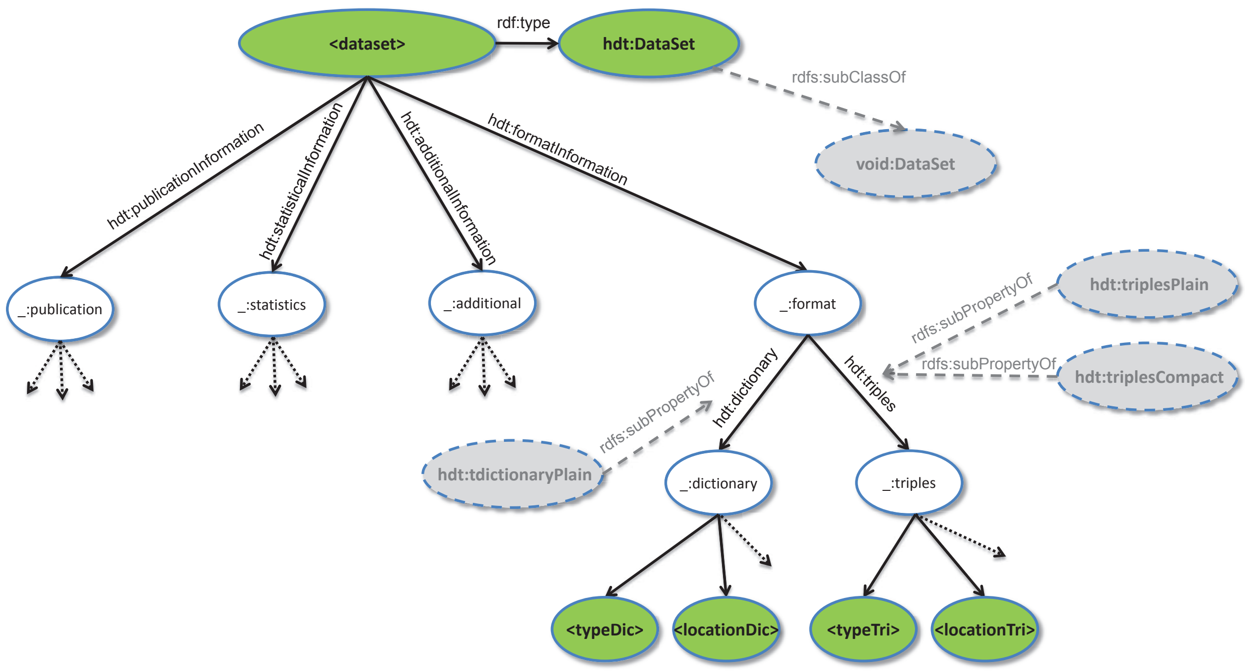


Figure 7.4: The structure of the proposed HDT practical deployment.

- **Publication Metadata** (`hdt:publicationInformation`), group the statements about the publication act (lines `16-22`). As can be seen in the example, it is strongly recommended to use standard vocabularies such as Dublin Core and FOAF. Additionally, VoiD properties can provide specific RDF features, such as the location of the associated SPARQL endpoint (line `22`).

- **Statistical Metadata** (`hdt:statisticalInformation`) include statistics such as the number of RDF triples of the dataset, or the number of different predicates. This is shown in lines `14-25`, exploiting VoiD properties. Note that statistics in VoiD are limited, hence a specific vocabulary could also include the metrics presented in Chapter 4. Other well-known vocabularies for statistics are encouraged, such as RDFStats (Langegger & Woss, 2009) for histograms, semantic statistics with SDMX (Cyganiak, Field, Gregory, Halb, & Tennison, 2010) or the RDF Data Cube Vocabulary (Cyganiak & Reynolds, 2013).

- **Format Metadata** (`hdt:formatInformation`) link to the concrete dictionary and triples encodings. The `hdt:dictionary` and `hdt:triples` properties group the metadata about the dictionary and triples respectively. Two mandatory properties are required to describe these components:

  – The specific **type** of the Dictionary and Triples implementations. This can be specified in two different ways:

    1. Stating the `rdf:type` of the component. In the header example, the line `30` states that the dictionary is of type `hdt:dictionaryPlain` (this is explained in the next section §7.2.2).
    2. Using RDFS subproperties of `hdt:dictionary` and `hdt:triples`, as shown in Figure 7.4. In the example, line `28` points to the `hdt:triplesCompact` configuration, implicitly denoting that the triples are in *Compact Triples* format (this is described in the following section, §7.2.3).

```
 1 @prefix void: <http://rdfs.org/ns/void#>.
 2 @prefix dc: <http://purl.org/dc/terms/>.
 3 @prefix foaf: <http://xmlns.com/foaf/0.1/>.
 4 @prefix hdt: <http://purl.org/HDT/hdt#>.
 5 @prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
 6 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
 7 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
 8 @prefix swp: <http://www.w3.org/2004/03/trix/swp-2/>.
 9
10 <http://example.org/myInfo.hdt> a hdt:Dataset;
11                               hdt:publicationInformation  _:publication;
12                               hdt:statisticalInformation  _:statistics;
13                               hdt:formatInformation       _:format;
14                               hdt:additionalInformation   _:additional .
15
16 _:publication   dc:issued "2013-01-01";
17                 dc:license <http://www.gnu.org/copyleft/fdl.html>;
18                 dc:publisher [  a foaf:Organization;
19                                 foaf:homepage <http://example.org/theCompany>];
20                 dc:source <http://downloads.example.org/1.0/en/>;
21                 dc:title "myInformation";
22                 void:sparqlEndpoint <http://example.org/myInfo/sparql> .
23
24 _:statistics    void:triples "7";
25                 void:properties "4" .
26
27 _:format        hdt:dictionary _:dictionary;
28                 hdt:triplesCompact _:triples .
29
30 _:dictionary    rdf:type hdt:dictionaryPlain;
31                 hdt:fileLocation <http://example.org/myInfo.hdt> ;
32                 dc:format "application/x-gzip";
33                 hdt:dictionaryEncoding "utf8";
34                 hdt:dictionaryNamespaces [hdt:namespace [hdt:prefixLabel "ex";
35                                                        hdt:prefixURI "http://example.org/"]];
36                 hdt:dictionaryOrder <hdt:alphabetical_order>;
37                 hdt:dictionarySeparator "\\0" .
38
39 _:triples       hdt:fileLocation <http://example.org/myInfo.hdt> .
40                 hdt:predicateStream [dc:format "application/octet-stream";
41                                       hdt:IDCodification hdt:logBits];
42                 hdt:objectStream    [dc:format "application/octet-stream";
43                                       hdt:IDCodification "32"];
44
45 _:additional    swp:signature "AZ8QWE..."^^<xsd:base64Binary>;
46                 swp:signatureMethod <swp:JjcC14N-md5-xor-rsa> .
```

Figure 7.5: A Header example in HDT.

- The **URI** to localize the dictionary and triples. If they are provided in the same file as the Header (in a standalone configuration), this URI will coincide with the current HDT URI. This is the case of our example, in which the URIs of the dataset in line 10 is equal to the URI of the dictionary, line 31, and triples, line 39. If the components are split in several chunks, one could make use of an RDF sequence (rdf:Seq) to number all the locations.

In turn, additional format metadata depend on the concrete implementation of both Dictionary and Triples components. Hence the metadata for our dictionary and triples practical approaches (lines 32-37 and 40-43) are detailed in the following sections. It is worth noting that all the additional metadata provided here are intended for discovering. The specific decisions for HDT decoding are delegated to specific control information described in the HDT syntax (§7.3).

- **Additional Metadata** (hdt:additionalInformation) gather all kind of additional information. Lines 44-46 provide a basic signature.

7.2.2 Plain Dictionary Encoding

We propose a *Plain Dictionary* by default, denoted in the Header with the `hdt:dictionaryPlain` type, an RDFS subproperty of `hdt:dictionary`. Plain dictionary is aimed at publishing and exchange, but it should contribute to an efficient parsing post-processing. That is, we must acknowledge that this plain dictionary is only a serialization which has to be loaded into some data structure in order to allow the minimum `locate` and `extract` operations at consumption time (see *Dictionary uses and operations for Consumption* in §7.1.2).

Similarly to any dictionary implementation, Plain Dictionary takes specific decisions for the mapping between RDF terms an IDs and its codification for serialization:

Mapping Function: We split the dictionary in the four common subsets commonly used by RDF engines (Atre et al., 2010), mapped as follows. Let us suppose 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 $[1, \ |P_G|]$.

Note that the subject-object ratio (Definition 18) characterizes the proportion of common subject-objects in the dictionary. The empirical study of this ratio already denoted a noticeable value of common subject-objects (see §4.3.1). Thus, the dictionary size is reduced versus a disjoint assignment of subjects and objects, because the common elements are encoded once. In addition, the set of predicates is treated independently because of their low number and the infrequent overlapping with other sets. This configuration minimizes the range of predicate IDs, hence it contributes to compactness in the triples substitution (smaller IDs are equivalent to less bits per ID).

An example of these four sets is shown in Figure 7.6, built upon the graph in Figure 4.1. One could argue that a potential ambiguity could be present when extracting the ID-triples. Note that an ID, such as `2` in the figure belong to different sets: *Subjects*, *Objects* and *Predicates*. However, the disambiguation is trivial as long as we know that the ID in a triple is acting as a subject, a predicate or an object (Fernández et al., 2013). Let us suppose that we are parsing and ID-triple such as $(2, 3, 3)$. The first ID-term is a subject, then it could be mapped whether in the *Common Subjects-Object* or the *Subjects* partition. As the maximum ID in *Common Subjects-Object* is 1, it is obvious that 2 belongs to the *Subjects* partition. An extract operation will return *<http://example.org/Javier>*. Next, the predicate in the triple is numbered as 3. As it is a predicate, it is unambiguously mapped in the *Predicate* partition, and then the `extract` operation retrieves *foaf:mbox*. Finally, the process runs similar for the object 3, retrieving “jfergar@example.org”.

Terms encoding: We assume an alphabetic order inside each set, and a sequential numeric mapping. The physical serialized data comprises a list of plain strings (typically in *utf8*) in order from (1) to (4). This is shown in Figure 7.7. We make use of a reserved character to delimit strings and sections. In particular, we reserve the '\0' ASCII character. A double '\0\0' denotes the end of dictionary section.

Finally, it is worth mentioning that one could modify these by-default parameters (such as the delimiting character), but it must be described by specific control information for the dictionary (described in the HDT syntax 7.3). These decisions can also be described in the HDT Header for discovering purposes. For instance, the example in Figure 7.5 describes the use of “utf8” encoding in terms (line `33`), the alphabetic order in each partition (line `36`) and the delimiting character (line `37`). In addition, we declare

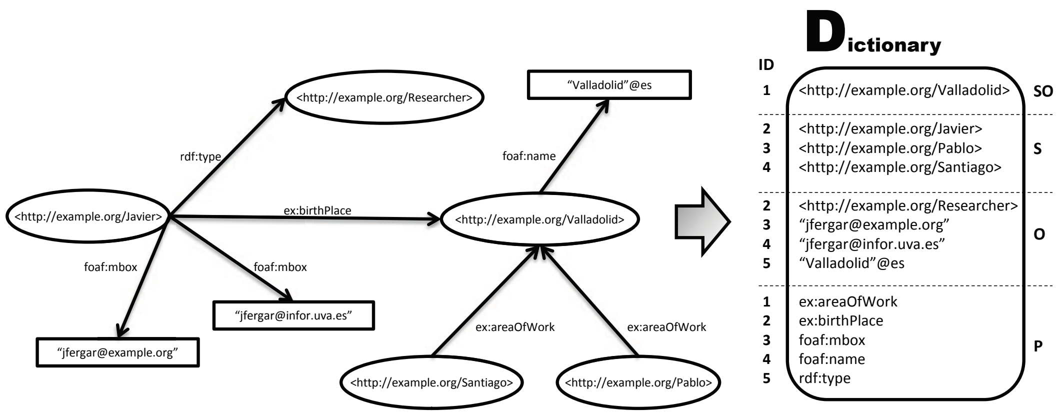

<http://example.org/Javier>
rdf:type
<http://example.org/Researcher>
ex:birthPlace
<http://example.org/Valladolid>
foaf:name
"Valladolid"@es
foaf:mbox
foaf:mbox
"jfergar@infor.uva.es"
"jfergar@example.org"
ex:areaOfWork
ex:areaOfWork
<http://example.org/Santiago>
<http://example.org/Pablo>
Dictionary
ID
1 <http://example.org/Valladolid> SO
2 <http://example.org/Javier>
3 <http://example.org/Pablo>
4 <http://example.org/Santiago> S
2 <http://example.org/Researcher>
3 "jfergar@example.org"
4 "jfergar@infor.uva.es"
5 "Valladolid"@es O
1 ex:areaOfWork
2 ex:birthPlace
3 foaf:mbox
4 foaf:name
5 rdf:type P

Figure 7.6: An example of the different sections in an HDT plain dictionary.

```
<http://example.org/Valladolid> \0\0 <http://example.org/Javier> \0 <http://example.org/Pablo> \0 <http://example.org/Santiago>
\0\0 <http://example.org/Researcher> \0 ''jfergar@example.org'' \0 ''jfergar@infor.uva.e'' \0 ''Valladolid''@es \0\0
ex:areaOfWork \0 ex:birthPlace \0 foaf:mbox \0 foaf:name \0 rdf:type \0\0
```

Figure 7.7: Serialized data of an HDT plain dictionary (from Figure 7.6).

a "ex" prefix (lines 34-35). Last, note that, in order to improve the final size for exchanging, all the dictionary stream is compressed with gzip (line 32).

7.2.3 Triples Encodings

We propose two simple encodings for the Triples component: *Plain Triples* and *Compact Triples*. Intuitively, both are aimed at compact serialization for exchange, thus it should be post-processed for consumption. Figure 7.8 illustrates *Plain* and *Compact Triples* over the example in Figure 7.6.

- **Plain Triples encoding** (`hdt:triplesPlain`) is the most basic approach. Plain Triples (PT) only exploits dictionary to perform the ID substitution of triples. Thus, the physical serialization contains three IDs per triple (shown in Figure 7.8(A)). In order to provide a certain order, the triples stream is sequentially sorted by subject, predicate and object IDs respectively. It is worth noting that one would make use of a number of fixed bits per ID (*e.g.* 32 or 64) or each ID can be encoded with $\log n$ bits, being n the number of total subjects, predicates or objects.

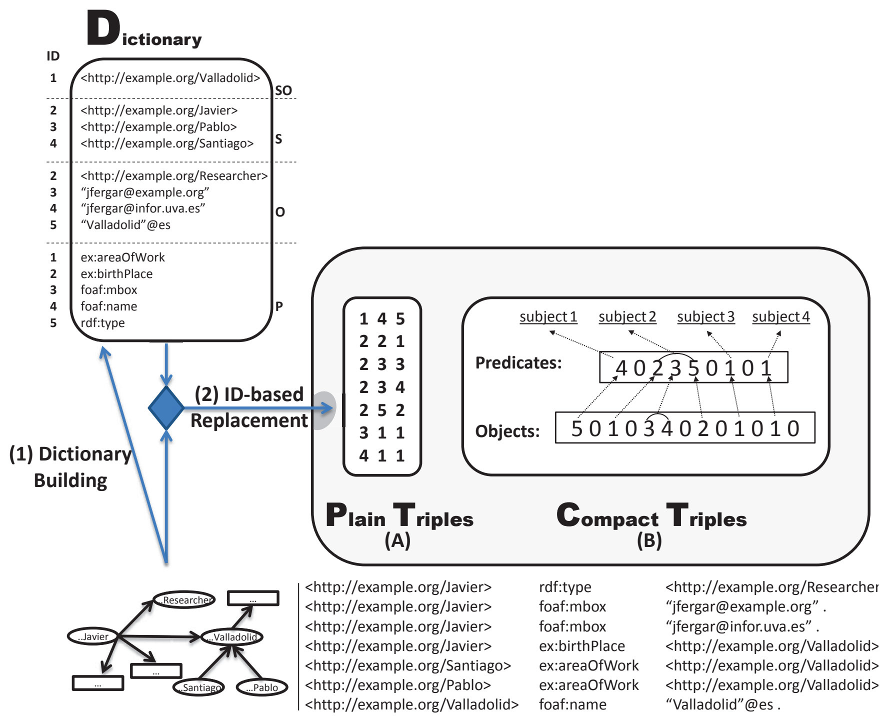
  These decisions must be specified in the specific control information for the triples (described in the HDT syntax 7.3), and can also be described in the HDT Header for discovering purposes.

- **Compact Triples encoding** (`hdt:triplesCompact`) reduces verbosity by creating adjacency lists in a similar way than N-Triples and Turtle do. As stated in Section 6.2.1, these syntaxes avoid repetitions, i) using a semicolon ";" to separate different predicates of the same subject, and ii) using a comma "," to separate different objects of the same pair of subject and predicate. We take the same underlying concept of adjacency list, though we overcome this idea taking advantage of the implicit order of the IDs. Let us consider the set of triples:

$$\{(s_1, p_1, o_{11}), \cdots, (s_1, p_1, o_{1n_1}), (s_1, p_2, o_{21}), \cdots (s_1, p_2, o_{2n_2}), \cdots (s_1, p_k, o_{kn_k})\}$$

  can be written then as the adjacency list (organized by subject):

$$s_1 \rightarrow [(p_1, (o_{11}, \cdots, o_{1n_1}), (p_2, (o_{21}, \cdots, o_{2n_2})), \cdots (p_k, (o_{kn_k}))].$$



<http://example.org/Javier>	rdf:type	<http://example.org/Researcher
<http://example.org/Javier>	foaf:mbox	"jfergar@example.org" .
<http://example.org/Javier>	foaf:mbox	"jfergar@infor.uva.es" .
<http://example.org/Javier>	ex:birthPlace	<http://example.org/Valladolid>
<http://example.org/Santiago>	ex:areaOfWork	<http://example.org/Valladolid>
<http://example.org/Pablo>	ex:areaOfWork	<http://example.org/Valladolid>
<http://example.org/Valladolid>	foaf:name	"Valladolid"@es .

Figure 7.8: Practical approaches for HDT triple serialization.

Assuming a set of subjects $S_G = \{s_1, s_2, \cdots, s_N\}$, the graph can be represented as all the adjacency lists of subjects:

$$
\begin{aligned}
&s_1 \rightarrow [(p_1, (o_{11}, \cdots, o_{1n_1}), (p_2, (o_{21}, \cdots, o_{2n_2})), \cdots (p_k, (o_{kn_k}))].\\
&s_2 \rightarrow [\cdots].\\
&\cdots\\
&s_N \rightarrow [\cdots].
\end{aligned}
$$

Compact Triples (CT) encodes these lists (a list of lists) compactly. First, note that, following the same sequential order as Plain Triples, subject IDs $s_1, s_2, \cdots, s_N$ are a correlative sequence. Thus, an immediate saving can be achieved by omitting the subject representation. In the final encoding, the first list corresponds to the first subject, the second list to the second subject, and so on. Next, the representation is slightly modified. We split the list of lists into two coordinated streams of `Predicates` and `Objects`, as shown in Figure 7.8 (B).

- The `Predicate stream` lists the predicates associated with subjects, maintaining the implicit grouping order. The end of a list of predicates is marked with the reserved zero ID[3]. In other words, predicate lists are separated by 0s, therefore the i-th list belongs to i-th subject.
- The `Object stream` lists the objects for each pair *(subject, predicate)*. In this case, the zero ID marks a change of*(subject, predicate)* pair, moving forward in the first stream processing. That is, the j-th list belongs to the j-th *(subject, predicate)* pair in the former stream.

This underlying representation entails several remarks:

- All predicates related to a subject are sorted in increasing way. For instance, in Figure 7.8, the predicates for the second subject are sorted as $\{$`2,3,5`$\}$. This is very similar to a well-known

[3]Note that the dictionary assigns IDs from 1.

problem: posting list encoding for information retrieval purposes (Baeza-Yates & Ribeiro-Neto, 2011; Witten, Moffat, & Bell, 1999).

– Objects are ordered for each pair *(subject, predicate)*. In our example, the object `5` is listed first (because it is related to the pair `(1,4)`), then `1` (related to `(2,2)`), next the objects `3,4` (by considering that both are related to the pair `(2,3)`), and so on.
– A Depth First Search (DFS) traversal of the forest retrieves all triples sorted by ID. That is, it obtains the Plain Triple list.

Similarly to PT, Compact Triples would make use of a number of fixed bits per ID (32 or 64 in practice) or each ID can be encoded according to the logarithm of the corresponding number of elements. CT includes two streams by default, hence different codifications can be used in each stream. Again, this must be specified in the specific control information for the triples (§7.3), and can also be described in the `HDT` Header for discovering purposes. For instance, the example in Figure 7.5 describes the use of log bits in the predicate stream (lines `40-41`) but a fixed number of 32 bits in the object stream (lines `32-43`).

Finally, it is very significant to note that some of the metrics proposed in Chapter 4 perfectly characterize both streams in CT. In short:

– The labeled out-degree of a given subject is the number of different predicates related to this subject. Thus, for every subject $s \in S_G$, the length of its list in the `Predicate stream` is exactly its labeled out-degree, $degL^-(s)$.
– In general, one could characterize the expected mean and maximum length of the lists in the `Predicate stream`, given by $\overline{degL^-}(G)$ and $degL^-(G)$ respectively.
– Symmetrically, the partial out-degree, $deg^{--}(s,p)$, denotes the size of the corresponding list in the `Object stream` for every valid pair $s \in S_G, p \in P_G$.
– In general, mean and maxim values of the lists in the `Object stream` are given by $\overline{deg^{--}}(G)$ and $deg^{--}(G)$ respectively.

7.3 `RDF/HDT` Syntax Specification

We have stated that `HDT` is flexible and provides multiple configurations for each of its three components, hence we provided a practical deployment for publication and exchange. In this section we summarize the `RDF/HDT` syntax specification to standardize the encoding of these multiple variations. Further details can be found in our W3C Member Submission (Fernández et al., 2011). Note that the W3C specifications were published in 2011. Thus, the following details of the syntax introduce some novel improvements which make the format slightly differ from the original approach (Fernández et al., 2011).

7.3.1 The Structure of an `HDT` File

Despite multiple configurations, `HDT` processors have to know how to manage `HDT` files. In other words, clear instructions on the structure of `HDT` files allow to implement an `HDT` encoder/decoder in any language/platform. An `HDT` file consists of the following items:

- One mandatory initial `Control Information` preamble.
- The `HDT` Header.
- Zero or more `HDT` Dictionary, each one preceded by a `Control Information`.
- Zero or more `HDT` Triples, each one preceded by a `Control Information`.

Cookie	Type	Codification	[Options]

Table 7.1: HDT Control Information.

Component Bits	Stands for
00	Global
01	Dictionary Component
10	Triples Component
11	Reserved

Table 7.2: Valid types in the HDT Control Information.

Thus, an HDT file must be headed by a Control Information preamble which establishes some general properties (described in the next section). It is worth mentioning that every Control Information is perfectly delimited as it starts with a “$HDT” keyword and ends with “$END” (both are reserved keywords). Thus, the HDT Header component, which provides metadata about the RDF dataset, is located right after the first Control Information preamble. The HDT Header is encoded in Turtle by default. Note that a void content could be provided (but not recommended).

Both the Dictionary and Triples components are optional. This feature may not be commonly used, but it allows to exchange only header information, which could be useful for discovering datasets. As stated, the HDT core data can be distributed in several chunks and under different formats. The metadata of the header could help retrieve the appropriate HDT core data.

Dictionary and Triples components are also preceded by a Control Information which can provide additional properties for each concrete implementation.

In the next sections we present the Control Information structure, and commonalities for every Dictionary and Triples implementations.

7.3.2 The Control Information

A Control Information (CI) is a preamble describing configuration options. It is used at the beginning of the HDT file as well as the Dictionary and Triples components. It has the following structure (showed in Table 7.1):

Cookie. The CI starts with an HDT Cookie, a magic keyword ’$HDT’, as four ASCII characters. These four bytes are particular to HDT and specific enough to distinguish HDT files and streams from a broad range of data types.

Type. The second part of the CI consists of two bits identifying the component or components that follow the CI, *i.e.* the component described by the CI. The valid values are provided in Table 7.2. A “00” value stands for the initial global preamble, whereas “01” and “10” indicate that the CI is describing a Dictionary or a Triples component respectively. We reserve the “11” value.

Codification. The third part of the CI identifies the codification being used in the following component (Dictionary or Triples). If the CI is the initial preamble of the file, this format indicates the HDT syntax version. The codification is given as a null-terminated string containing a URI of the concrete implementation. This way i) we leverage the same URI infrastructure and ii) remain flexible for future codifications. Table 7.3 shows the default URIs for the aforementioned practical deployments. Note that for the global description, hdt:HDTv0.9 stands for this current version of the RDF/HDT syntax. Thus, future version could be added by defining the appropriate URI.

HDT Practical Component	Reference URI
current RDF/HDT syntax	hdt:HDTv0.9
Plain Dictionary	hdt:dictionaryPlain
Plain Triples	hdt:triplesPlain
Compact Triples	hdt:triplesCompact

Table 7.3: Reference URIs of the HDT practical components.

Options. The last part of the CI provides a mechanism to specify additional properties of the global HDT file or the concrete Dictionary or Triples component. Properties are ASCII strings with the scheme:

$$<\text{property}_1>=<\text{value}_1>;<\text{property}_2>=<\text{value}_2>;\cdots<\text{property}_N>=<\text{value}_N>;\backslash 0$$

noting that the list of properties is finished by a NULL character and, obviously neither properties nor values can include "=" nor ";" symbols.

These auxiliary properties are used to provide the necessary information to process the data. We reserve one property, `format`, as a standard property to identify the MIME type of the Header (in case of a global CI), Dictionary or Triples component. This is the property, for instance, in which we set up the concrete RDF syntax used in the Header.

Finally, a reserved word "$END" must be added at the end of the CI to delimit its length.

7.3.3 Plain Dictionary Encoding

Plain Dictionary encoding follows the description of Section 7.2.2. As stated, the serialization consists of a plain bulk of the strings in each dictionary section, with a reserved separator between them. This was shown in Figure 7.7. We provide additional remarks to complete a standard serialization.

Strings encoding. Plain Dictionary follows the N3 syntax for the RDF terms, *i.e.*, to distinguish between URIs, literals and blank nodes (Fernández et al., 2013):

- URIs are delimited by angle brackets "<" and ">".
- URIs can be absolute or relative to the base URI (defined as a property in the CI of the Dictionary component).
- URIs can make use of prefixes (defined as a property in the CI of the Dictionary component) or predefined prefixes (described below).
- Blank nodes are named with the _: namespace prefix, *e.g.* _:b83 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 representing 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 7.4 shows the predefined prefixes whereas Table 7.5 describes the string escaping sequences which follows N3.

	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 7.4: Dictionary predefined prefixes.

	Stands for
`\newline`	Ignored
`\\`	Backslash (\)
`\'`	Single quote (')
`\"`	Double quote (")
`\n`	ASCII Linefeed (LF)
`\r`	ASCII Carriage Return (CR)
`\t`	ASCII Horizontal Tab (TAB)
`\uhhhh`	character in BMP with Unicode value U+hhhh
`\U00hhhhhh`	character in plain 1-16 with Unicode value U+hhhhhh

Table 7.5: Dictionary string escaping sequences.

Property	Use
dictionaryEncoding	Set up the dictionary encoding. By default, utf8.
dictionarySeparator	Define the reserved separator character. As stated, the default value is "\0".
dictionaryOrder	Describe the order inside each defined subset in the dictionary.
PrefixBaseURI	Set up the base prefix to be used in the dictionary when parsing relative URIs.
PrefixLabel_1, PrefixLabel_2, etc.	Set up prefixes labels to be used in the dictionary.
PrefixURI_1, PrefixURI_2, etc.	Set up the corresponding URIs to the predefined prefix labels.
Sequence	Identify the order in the sequence of all the dictionary chunks in case of splitting.

Table 7.6: Plain Dictionary properties in the Control Information.

Properties in Control Information. As stated above, both Base URI and user-defined prefixes can be established. These and other decisions of encoding are provided with properties in the CI of the dictionary section. Table 7.6 shows the possible parameters for a Plain Dictionary. For instance, if present, the *dictionaryOrder* property establishes the order of the mapping within each of the four Plain Dictionary sections. By default the order is "alphabetic", but other accepted values are "none" and "frequency". In the latter case, terms are ordered by number of occurrences within the triples.

7.3.4 Triples Encodings

In Section 7.2.3 we proposed two practical encodings for the Triples component: *Plain Triples* and *Compact Triples*. The concrete encoding of an `HDT` dataset is established in the `codification` value of the Control Information. The reference URI of each proposal is given in Table 7.3.

Properties in Control Information. Obviously, the properties depends on the concrete triples approach. Nevertheless, both encodings share a set of common parameters which can be defined its corresponding CI. These parameters are summarized in Table 7.7.

Plain Triples. It follows the same simple notions provided in Section 7.2.3. The final physical serialization contains a continuous streams of IDs, with three IDs per triple. In addition to the properties in Table 7.7, it must provide the number of bits per element in the CI:

- `IDCodificationBits:` establishes the number of bits per ID. One could expect a number (the default value is 32 bits) or the URI `hdt:logBits` denoting that each ID is encoded with

Property	Use
Triples	Indicate the total number of triples.
Order	Set up the triples ordering: SPO (default), SOP, PSO, POS, OPS, OSP or none.
Subjects, Predicates and Objects	Provide the number of different elements respectively.
Sequence	Identify the order in the sequence of all the triples chunks in case of splitting.

Table 7.7: Common triple properties in the Control Information.

$\log(n)$ bits, being n the number of total subjects, predicates or objects. These numbers must then be provided in the CI (as shown in Table 7.7).

Compact Triples. The CT encoding splits the representation into two streams of IDs (Predicates and Objects in case of a SPO order). The streams are encoded one after the other. In order to know the limit and bits per ID of each stream, both Compact and Bitmap Triples make use of additional properties in the CI.

- `FirstStreamLength:` indicates the number of elements in the first stream.
- `SecondStreamLength:` indicates the number of elements in the second stream.
- `FirstIDCodificationBits:` provides the number of bits of each ID in the first stream. The default value is set to 32. The value logBits must be interpreted as follows: each ID is encoded with $\log(n)$ bits, being n the number of the elements in this stream.
- `SecondIDCodificationBits:` provides the number of bits of each ID in the second stream, with the same policy than the previous property.

7.4 Experimental Evaluation

This section evaluates the size and performance of the practical `HDT` deployment for publication and exchange presented in the previous Section 7.2.

First, we measure the size of the `HDT` Dictionary and Triples to show its compactness (§7.4.1). Then, we evaluate the scalability of `HDT` based on the implementation of Plain Dictionary and Compact Triples (§7.4.2). Finally, we perform and evaluate traditional compression on top of an `HDT` dataset (§7.4.3).

This experimentation runs on the datasets described in Chapter 4, Section 4.2. For the evaluation, we consider a Header in Turtle syntax such as the one in Figure 7.5. Note that the size of the Header (a few KB at most) is negligible at large scale.

As this section studies the Publication-Exchange workflow, we design a real-world setup in which two main stakeholders are involved (Table 7.8 details their characteristics):

- The **data publisher** is implemented on a powerful computational configuration. It simulates an efficient data provider within the Web of Data.
- The **consumer** is designed on a configuration able to play the role of an agent consuming Big Semantic Data. Thus, we assume a powerful computational configuration, although slightly more limited than the data publisher.

Finally, let us remark that we use a g++ 4.7.2 compiler with `-O9` optimization for all the tools, which are openly provided at `http://rdfhdt.org`.

Machine	**Data publisher**	**Consumer**
Num. of CPUs	4	8
CPU	Intel Xeon X5675	Intel Core i7 3820
CPU speed	3.07 GHz	3.6 GHz
cache size L1/L2	1 MB / 256 KB	64 KB / 256 KB
RAM size	48 GB	16 GB
I/O cached reads:	7,200 MB/sec	13,100 MB/sec
I/O buffered disk reads:	190 MB/sec	194 MB/sec
RAID disks	8 of 1TB, SAS 7,200 RPM	1 of 1TB, SATA 7200 RPM
RAID level	10	-
Operating System	Ubuntu/Precise 12.04.2 LTS	Debian 7.1

Table 7.8: Machines configuration of the experimental framework.

7.4.1 Dictionary and Triples Compact Ability

Table 7.9 shows the compact ratios of each proposed component in `HDT` with respect to the original N-Triples format (one triple per line). For the sake of clarity, we present the datasets in ascending order of triples. Plain Triples (PT) and Compact Triples (CT) are represented according to the logarithm of the corresponding number of elements (see §7.2.3).

First of all, it is remarkable that in all datasets, except for the *2000 US Census* commented below, the size of the Dictionary is significant bigger than the corresponding size for Triples (both in PT and CT). Whereas the size of the Plain Dictionary is around 12% the size of the original dataset, and up to 21%, Plain and Compact Triples are all in the range 2.5% - 5.6%. In some cases, such as *Jamendo*, *SWDF* or *DBLP*, the dictionary is 6 times bigger than the triples. This first result points to the need of improving the representation of both components to boost the final compression result. These insights encourage the design of compact but functional Dictionaries and Triples that we address in Part III and IV respectively.

In addition, Table 7.9 shows that PT and CT have a comparable ratio. Nevertheless, as we expected, Compact Triples outperforms Plain Triples in all datasets. The only exception is again the *2000 US Census*. Note that this dataset includes a particular structure in which almost all subjects make use of shared blank nodes to organize the different types of census figures or measures. In this scenario, it is possible that i) the triple structure exceeds the dictionary size, as there is a low number of different values and ii) PT outperforms CT, as the adjacency lists are too short and CT pays the overhead of the delimiting character of each list. Nevertheless, this is a corner case and the final compression ratio figures remain very close.

Next, Table 7.10 compares the compression ratio of `HDT` with Plain and Compact Triples against three well-known universal compressors. We choose `gzip`[4] and `lzma`[5] as two dictionary-based techniques on Lempel-Ziv compression, and `bzip2` based on the Burrows-Wheeler Transform.

The most effective universal compressors for all datasets are `bzip2` and `lzma` which achieve ratios of around 4%. Note that the `HDT` representation is completely "in plain". That is, these results are obtained by representing the dictionary and triples components aside, grouping references in the dictionary, using a *log* bits (of the corresponding number of elements) codification in ID-triples, and using adjacency lists (in CT). In other words, with these simple decisions, compression ratios are around 16%, only 2 times bigger (on average) than a pure data compression with gzip. This demonstrates the previously cited ability of `HDT` to obtain compact representations of RDF.

Nonetheless, we present below an additional compression on top of `HDT` (§7.4.3), which can fit in very strict exchanging scenarios requiring even a better compression than traditional compressors.

[4] `http://www.gzip.org`
[5] `http://www.7-zip.org`

Dataset	Original Size (MB)	Plain Dictionary	Triples	
			Plain	Compact
SWDF	16	15.26%	2.93%	2.65%
2011 Australian Census	52	5.88%	2.82%	2.63%
Jamendo	144	21.13%	3.73%	3.51%
AEMET	726	11.20%	2.56%	2.49%
LinkedMDB	850	11.66%	4.22%	4.12%
Wordnet	974	9.10%	3.75%	3.57%
Affymetrix	6,526	11.97%	4.20%	3.67%
Flickr	6,714	12.05%	4.53%	3.80%
Dbtune	9,566	10.39%	4.19%	4.02%
DBLP	9,799	16.82%	3.80%	3.32%
2000 US Census	21,796	2.63%	4.81%	4.87%
Linked Geo Data	39,423	21.49%	5.58%	5.47%
Dbpedia 3-8	63,053	12.78%	5.55%	3.77%
Ike	102,662	8.40%	3.47%	3.31%

Table 7.9: Compression ratio of the Dictionary and Triples components with respect to the original size of each dataset.

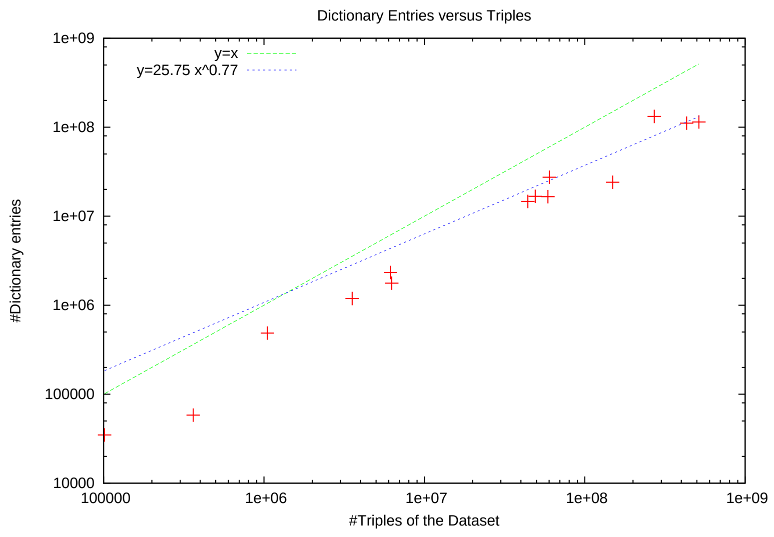


Figure 7.9: HDT dictionary growth with respect to the number of triples in the dataset. Both axes are drawn in logarithmic scale.

7.4.2 Scalability Evaluation

We evaluate the HDT scalability in three correlated aspects: dictionary size, HDT compact ability and performance (at publisher and consumer). First, we study the **dictionary growth** with respect to the number of triples. Note that the dictionary is seen as the largest HDT component, as shown in Table 7.9.

Figure 7.9 represents (in logarithmic scale) the number of entries of the dictionary versus the number of triples of the dataset. Each point corresponds to one of the 14 different evaluation datasets. We consider a Plain Dictionary (§7.2.2), hence the "number of entries" is the sum of all the elements in each subdivision: *common subject-objects*, *non common subjects*, *non common objects* and *predicates*.

Note that Figure 7.9 also represents the $y = x$ function and the adjusted function fitting the distribution, $y = 25.75x^{0.77}$. As can be seen, the number of unique dictionary entries has a sublinear growth

Dataset	Triples (millions)	Size (MB)	HDT PT	HDT CT	Universal Compressors gzip	bzip2	lzma
SWDF	0.1	16	18.21%	17.92%	9.68%	6.63%	7.03%
2011 Australian Census	0.4	52	8.70%	8.51%	2.80%	1.33%	1.85%
Jamendo	1.0	144	24.87%	24.64%	5.83%	4.16%	4.00%
AEMET	3.5	726	13.77%	13.69%	2.57%	1.20%	1.37%
LinkedMDB	6.1	850	15.89%	15.79%	4.75%	2.79%	3.23%
Wordnet	6.3	974	12.85%	12.66%	4.97%	3.22%	4.32%
Affymetrix	44.2	6,526	16.17%	15.64%	5.42%	3.43%	3.91%
Flickr	49.1	6,714	16.58%	15.84%	9.03%	7.40%	6.28%
Dbtune	58.9	9,566	14.57%	14.41%	11.24%	7.65%	5.98%
DBLP	60.1	9,799	20.62%	20.14%	5.42%	3.49%	4.59%
2000 US Census	149.2	21,796	7.45%	7.50%	4.62%	2.27%	2.83%
Linked Geo Data	274.7	39,423	27.07%	26.96%	5.90%	4.13%	4.39%
Dbpedia 3-8	431.4	63,053	18.32%	16.55%	8.01%	5.90%	6.17%
Ike	514.8	102,662	11.86%	11.71%	3.22%	1.08%	1.50%

Table 7.10: Compression ratio of HDT with Plain and Compact Triples and universal compressors.

w.r.t. the number of triples. This result points that we can assure that the appropriate treatment in HDT can maintain the sublinear tendency in size, guaranteeing the scalability of the representation.

Next, we study the **HDT compact ability with incremental sizes**. To do so, we test the compression ratios of the Ike dataset (see description in Section 4.2), incrementally split in steps of 50 million triples up to the total size of the dataset (515 M.). We choose this particular dataset because it includes similar meteorological measures in different days. This way, we assure that incremental sizes actually share the same data modeling and identical properties. This gives the opportunity to carry out a precise evaluation of the evolution of the size in HDT.

The evaluation is shown in Figure 7.10. The top table studies the HDT evolution of effectiveness. As can be seen, the compression ratios for Plain and Compact Triples are in tune with the previous results. This ensures HDT effectiveness by considering that the effectiveness is achieved regardless the size of the dataset. Moreover, we can observe that the ratios decrease as the number of triples increases, going between 15% for 50 M. triples to around 12% for the 515 M. dataset. This can be seen as a natural reflection of the sublinear tendency of the dictionary growth. In other words, for increasingly large datasets, the proportion of new entries tends to decrease (the dictionary contributes to the total HDT size in less proportion), and thus more compression ratios can be achieved. The two rightmost columns of the top table show the memory usage for the creation of HDT (PT and CT figures are comparable). We provide the memory peak usage in GB and the ratio over the original size. It is worth noting that the creation process always employs less than 40% of the original size. It also follows a decreasing tendency in accordance to the number of triples. For the complete dataset, only 1/3 of the original size is used, resulting in a highly scalable process.

The bottom graph on Figure 7.10 shows the **HDT creation times in the publisher machine**. In this scenario, 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 linear growth. Note that both Plain and Compact Triples configurations provide comparable times. Nonetheless, CT remains below PT times: as less information is managed in CT, it requires less transfers to disk for the final dump of the representation.

Finally, we study the **HDT performance at the consumer**. This evaluation is done on incremental sizes of the *2000 US Census* (see description in Section 4.2), in steps of 15 million triples up to the total size of the dataset (150 M.). We choose this particular dataset because it provides similar features as

Triples (millions)	Size (GB)	HDT PT	HDT CT	Memory Peak Usage (creation) Size (GB)	Memory Peak Usage (creation) % mem/original
50	9	15,68%	15,37%	3,6	38,17%
100	19	15,91%	15,54%	6,9	36,18%
150	28	15,88%	15,45%	10	35,13%
200	38	15,33%	15,22%	13	34,02%
250	48	14,46%	14,35%	15	31,28%
300	58	13,88%	13,82%	19	32,71%
350	68	13,52%	13,52%	23	33,73%
400	78	13,16%	13,17%	25	32,02%
450	88	12,89%	12,89%	27	30,78%
500	97	12,11%	12,00%	30	30,80%
515	100	11,86%	11,71%	32	31,92%

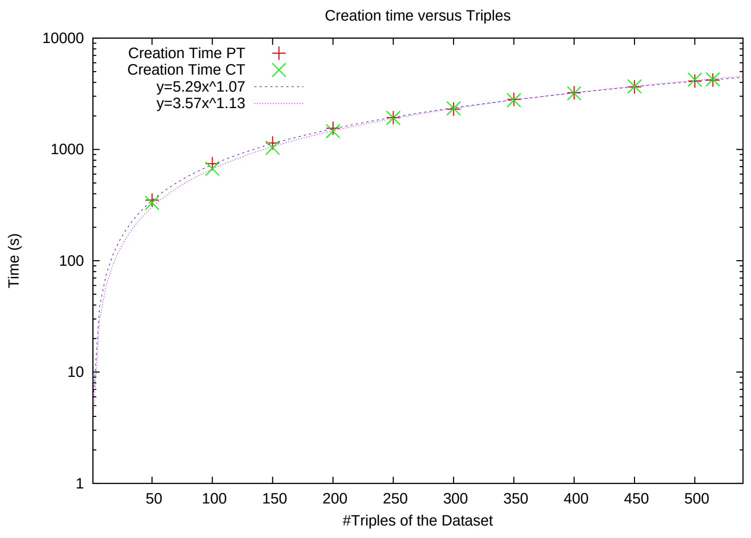


Figure 7.10: Performance of HDT (Plain and Compact) with incremental size from *Ike*. The top table shows effectiveness, whereas the bottom figure draws creation times.

the previous meteorological case (the structure remains similar for increasing number of triples) and it perfectly fits in a typical client such as the proposed in our evaluation framework (§7.4).

Figure 7.11 shows the results of this evaluation. The top table represents the HDT Plain Triples effectiveness in space, the memory used for creation at publisher, and for loading at consumption. It is important mentioning that, in this case, “*loading*” means to retrieve the dictionary and triples components of the HDT representation and to load them in memory structures, being functional for basic consumption. In this test, we load each of the four sets in Plain Dictionary into a Hash structure (hence we compute its overhead in size), and the Plain Triples in a sorted array. We obviate CT comparison for the sake of clarity as it provided very close results.

As can be seen, both the HDT PT size and the memory peak usage at publisher follow a similar tendency to that observed for *Ike*: the compression and memory usage ratio decrease as the number of triples increases. In addition, we can observe an identical tendency in the use of memory at loading. For the full dataset, the memory usage in the consumer is slightly above 20% of the original N-Triples size. Note that this size computes all the aforementioned structures in memory, required for RDF retrieval.

In turn, the bottom graph on Figure 7.11 shows the creation and loading times for HDT, in the publisher and consumer machines respectively. As in the previous case of *Ike*, the *creation* time follows

Triples (millions)	Size (GB)	HDT PT	Memory Peak Usage (creation)		Memory Peak Usage (load)	
			Size (GB)	% mem/original	Size (GB)	% mem/original
15	2	9.77%	0.87	36.39%	0.86	35.97%
30	5	8.15%	1.4	30.10%	1.3	27.95%
45	7	7.80%	1.9	28.07%	1.8	26.60%
60	9	7.29%	2.5	27.27%	2.1	22.91%
75	11	7.25%	3.1	27.40%	2.6	22.98%
90	13	7.34%	3.7	27.71%	3.2	23.97%
105	15	7.26%	4.1	26.52%	3.5	22.64%
120	17	7.45%	4.6	26.31%	4	22.88%
135	19	7.43%	5.1	26.17%	4.3	22.06%
150	21	7.39%	5.5	25.63%	4.7	21.90%

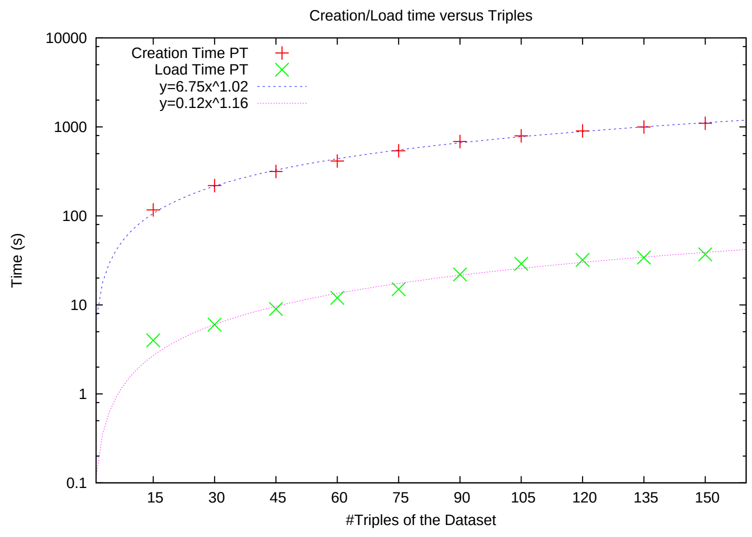


Figure 7.11: Performance of HDT (Plain) with incremental size from the *2000 US Census*. The top table shows effectiveness, whereas the bottom figure draws creation (publisher) and load times (consumer).

a linear growth, which is also replicated for the *loading* time: as can be seen, the loading time is only a very small fraction ($\approx 3\%$) of the creation one. Note again that the creation phase is made once at consumption, whereas loading is made in every potential consumer.

7.4.3 Additional HDT Compression

HDT achieves a significant reduction of the RDF dataset size by means of the Plain Dictionary and the Plain or Compact Triples configurations. This provides a clean publication scheme together with efficient compression ratios. However, we have stated that traditional compression outperforms the size of this representation. Obviously, this reduction is at the cost of decompression at consumption time (which can be very significant for techniques such as bzip2 and lzma). Moreover, data after decompression remain in the same plain RDF format (such as N-Triples).

Thus, we state that HDT can be even more compressible with little effort, fitting very strict exchanging scenarios. We test HDT compressibility with a particular deployment called HDT CT-Compressed. This deployment simply applies a gzip compression on the HDT dataset in Compact Triples.

Table 7.11 shows the results of HDT CT-Compressed with respect to the traditional gzip and bzip2 compression (over the original N-Triples). As can be seen, HDT CT-Compressed achieves the

Dataset	Size (MB)	`HDT`		Universal Compressors	
		`CT`	`CT-Compressed`	`gzip`	`bzip2`
SWDF	16	17.92%	5.67%	9.68%	6.63%
2011 Australian Census	52	8.51%	0.80%	2.80%	1.33%
Jamendo	144	24.64%	4.15%	5.83%	4.16%
AEMET	726	13.69%	1.03%	2.57%	1.20%
LinkedMDB	850	15.79%	2.35%	4.75%	2.79%
Wordnet	974	12.66%	2.27%	4.97%	3.22%
Affymetrix	6,526	15.64%	2.44%	5.42%	3.43%
Flickr	6,714	15.84%	3.57%	9.03%	7.40%
Dbtune	9,566	14.41%	2.58%	11.24%	7.65%
DBLP	9,799	20.14%	3.52%	5.42%	3.49%
2000 US Census	21,796	7.50%	1.30%	4.62%	2.27%
Linked Geo Data	39,423	26.96%	3.70%	5.90%	4.13%
Dbpedia 3-8	63,053	16.55%	4.64%	8.01%	5.90%
Ike	102,662	11.71%	0.78%	3.22%	1.08%

Table 7.11: Compression results of a gzipped `HDT` Compact Tiples representation (`HDT CT-Compress`).

most effective results with ratios between $2-4\%$ for all the considered datasets (except for a slight difference in *DBLP*). 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, improving the best `bzip2` results a mean of 25%.

These results show that `HDT` and its subsequent compression arises as the most efficient choice for exchanging RDF within the Web of Data. In the next parts we focus on making the exchanged datasets queryable for consumption.

- Why, you wouldn't follow that beast alone? - Someone's got to stay on his trail while it's hot!

King Kong (1933)

8 Discussion

In this part of the thesis we have addressed the scalable publication and exchange of Big Semantic Data. This chapter ends this part with a brief summary illustrating our main contributions (§8.1) and a compact overview of the next steps (§8.2), retaken in the following parts of this thesis.

8.1 Contributions

We started this part of the thesis, in Chapter 6, with an introduction to the scalability drawbacks arising in Big Semantic Data. We developed a simple classification on the main different stakeholders acting in the current Web of Data. Although this categorization may be extended to cover all corner cases, it is a first step in the identification of the roles, natures, and different scalability problems of the stakeholders.

We then presented and characterized a common Publication-Exchange-Consumption workflow taking part in almost every application in the Web of Data. After reviewing the state of the art, we stated that these processes (and their stakeholders) are compromised at large scale by plain, non-functional, human-readable formats while managing Big Semantic Data. In short, we argued that they are very verbose and space-inefficient, they obviate metadata and other facilities to upgrade publication and enable discovery for consumption, and more importantly, they do not provide any means of direct access to the data.

These problems motivated the need of an efficient machine-processable RDF representation, addressed in Chapter 7. In this Chapter, we proposed `HDT`, a binary serialization format for RDF publication and exchange at large scale, and the basis for direct consumption (addressed in the following parts of this thesis).

We first described the conceptual *philosophy* of the `HDT` components (Header, Dictionary and Triples). We provided the definition of each flexible component, detailing their different operations and intended use. Then, we instantiate a concrete practical deployment of `HDT` with a *Plain Dictionary* encoding and two simple encodings for the Triples: *Plain Triples* and *Compact Triples*. In turn, we developed the `RDF/HDT` syntax specification as a well-defined but flexible container of `HDT-based` datasets.

Finally, we performed a deep evaluation which analyzes `HDT` features on real-world datasets. Main conclusions can be summarized as follows:

- The size of the dictionary $(12 - 21\%)$ is significant bigger than the corresponding size for Triples $(2.5 - 5.6\%)$. This pushes the need of addressing both components (addressed in Part III and IV) to enrich the final representation.
- The mere `HDT` decomposition leads to large space savings: This simple decision takes around 16% of the original representation (N-Triples) and only 2 times more space than a gzip compression.
- `HDT CT-Compressed`, a particular deployment which applies a gzip compression on the `HDT` dataset in Compact Triples, outperforms traditional compression (including gzip). It improves the `bzip2` results a mean of 25%.

- Our study of scalability shows that the number of unique dictionary entries has a sublinear growth *w.r.t.* the number of triples. We also report that compression ratios remain high at incremental sizes, guaranteeing the scalability of the representation.
- Both the creations and loading of HDT are highly scalable processes: the creation performance employs less than 40% of the original size for the considered datasets, following a decreasing tendency *w.r.t* the number of triples. The memory usage for loading in the consumer can be estimated in 1/3 of the original size, with a similar decreasing tendency.
- The creation and loading times follow a linear growth, and the loading time is only a very small fraction ($\approx 3\%$) of the creation one.

These results 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.

8.2 Next Steps

HDT is designed as a binary RDF format to fulfill the requirements of portability (from and to other formats), clear publication scheme, compact ability, parsing efficiency (readiness for post-processing) and direct access to pieces of data in the dataset.

In the next parts of this thesis, we argue that HDT-encoded datasets can be directly consumed. We will show that lightweight indexes can be created once the different components are loaded into the memory hierarchy at the consumer. Thus, more complex operations can be achieved almost directly on the exchanged HDT datasets. This positions HDT as an integrated solution to manage Big Semantic Data in a Publication-Exchange-Consumption workflow.

Part III

Compressed Rich-Functional RDF Dictionaries

The world is not in your books and maps.
It's out there.

The Hobbit: An Unexpected Journey
(2012)

9 Introduction

We start a new part of the thesis, specifically focused on RDF dictionaries. This chapter motivates the need of advanced RDF dictionaries when managing Big Semantic Data (§9.1). As RDF dictionaries could be seen as a particular case of string dictionaries, we review different techniques for compressed dictionaries of general strings (§9.1). We also study the specific RDF dictionaries used in the Web of Data (§9.3). Finally, we list our future goals (§9.4) which concern the adaptation of the former techniques to provide specific and scalable RDF dictionaries.

9.1 Motivation

The previous Chapter 7 presented the notion of RDF dictionary. Rephrasing the definition, an RDF dictionary is a bijective function, $D : string \rightarrow ID$, which maps the strings representing the terms and the integer values (IDs) which identify them. Then, all triples in the dataset can be rewritten by replacing the terms with their corresponding ID.

Later in Section 7.2.2, we proposed a Plain Dictionary encoding showing high compression ratios. This approach, though, is a serialization aimed at exchange and it demands additional structures to be functional at consumption. As we stated, *"it has to be loaded into some structure (hash, B-trees) in order to allow searches (locate, extract, etc.) at consumption time"*.

In particular, a functional dictionary for consumption must provide two complementary operations (detailed in Section 7.1.2): (i) the *string-to-ID* operation, `locate(term)`, which returns the ID of a given term, and (ii) the *ID-to-string*, `extract(id)`, which retrieves the term identified by a given ID.

When most query processors perform on the ID-triples representation (Neumann & Weikum, 2010), both operations are exhaustively used by SPARQL engines during the query resolution process. Let us consider a SPARQL *triple pattern, (x,y,z),* in which `x`, `y`, or `z` may be a term in the RDF graph or a variable. Thus, the engine proceeds as follows:

1. It makes use of the dictionary to **locate** the IDs associated to the terms provided in the SPARQL triple patterns.
2. It transforms the given triple pattern of strings into a triple pattern of IDs.
3. It searches the pattern into the ID-triples representation, where the resulting ID values are bound to the variables given in the query.
4. It **extracts** the terms associated to these bounded IDs and returns the resulting subset of strings.

Note that, for SPARQL querying, `extract` is used many times as results are returned for each variable in the query, whereas the use of `locate` is limited to the number of terms bounded in the query. In this scenario, `extract` is overused in comparison to `locate`.

Most semantic applications implementing SPARQL are well-founded on a similar scenario, hence a functional dictionary is highly exploited in consumption processes. In addition, the dictionary could be

Technique		**Operations**	**Scenario**	**Stand out**
`Hash`	*(Hashing)*	`L,E`	General	Fast locate
`PFC`	*(Front-Coding)*	`L,E, pref`	Repeated prefixes	Tradeoff space/**time**
`HTFC`	*(Front-Coding)*	`L,E, pref`	Repeated prefixes	Tradeoff **space**/time
`Re-Pair`	*(Grammar-Comp.)*	`L,E, pref`	Repeated substrings	Tradeoff **space**/time
`FM-Index`	*(Self-Indexing)*	`L,E, substr`	General	Broad functional coverage

Table 9.1: Techniques for compressed string dictionaries: `L,E` stand for `locate` and `extract` respectively; `pref` and `substr` denotes support for prefix and substring locates.

used to resolve more specific matchings like the required for *filtering*. This is an interesting challenge by considering that an early `FILTER` evaluation allows query performance to be improved when the space of RDF triples to be explored is considerable reduced (Schmidt, Hornung, Lausen, & Pinkel, 2008).

However, the use of functional RDF dictionaries for consumption is also compromised in Big Semantic Data. The space required by the dictionaries is even larger than that used for the resulting ID-triples representations (as showed in Section 7.4.1). Whereas specific ID-triple indexes have been proposed for RDF (detailed in Section 6.2.1), specific RDF dictionaries are not fully addressed to the best of our knowledge. In other words, RDF stores currently make use of classical approaches for string dictionaries, and they do not scale (Brisaboa, Cánovas, Claude, Martínez-Prieto, & Navarro, 2011).

These classical techniques suffer from scalability issues. *Hashing*, for instance, holds plain strings and hence it dissuades applications handling the large vocabularies contained in Big Semantic Data. The use of *B-tree* (Bayer & McCreight, 1970) based solutions is the alternative, considering their optimization for large scale disk representations. However, the efficiency is compromised by the I/O costs derived from disk transfers.

In this scenario, *compression* arises as the natural solution for increasing the amount of data which can be efficiently managed in memory. This fact was already pointed out by Hogan (2011), when claims that a dictionary of URIs (for a web reasoning application) requires a prohibitive amount of memory to be stored and its compression would help increase the in-memory capacity.

Next section revises different approaches for compressed dictionaries of general strings. Then, we review the state of the art for RDF dictionaries. Finally, we describe the objectives of our compressed rich-functional RDF dictionary for Big Semantic Data.

9.2 Compressed String Dictionaries

RDF terms consists of elements from the vocabulary of Uniform Resource Identifiers (*URIs*), blank nodes, and literals. As all three can be seen as strings, the complete term collection (referred to as *vocabulary*) can be mapped as a traditional string dictionary.

String dictionaries (such as hashing or B-trees) are, in fact, the natural precedent of RDF dictionaries. Their conception and basic functionality is actually similar. A string dictionary $\mathcal{D}$ holds an ID-mapping of all different strings $\{s_1, s_2, \ldots, s_n\}$ used in a dataset (*vocabulary*), providing the operation:

- `locate(`s_i`)` which maps the string s_i into its ID in $\mathcal{D}$.

Typically, an additional structure must be implemented on top of $\mathcal{D}$ to provide the reverse operation:

- `extract(i)` which returns the string s_i identified as i in $\mathcal{D}$.

Compressed string dictionaries (Brisaboa et al., 2011) introduce compression and succinct data structures to lightweight scalability issues of string dictionaries, remaining efficient in performance.

An initial work by Bender, Farach-Colton, and Kuszmaul (2006) starts proposing a variant of the B-tree technique. They develop a cache-oblivious tree in which leaves are compressed with a technique called Front-Coding (Witten et al., 1999), described in Section 9.2.2. Later, this approach was improved by the compressed *permuterm* (Ferragina & Venturini, 2010). The original *permuterm* (Garfield, 1976) augmented each term with various rotations of its characters, resolving pattern queries with one wild-card symbol. The compressed version is a space-efficient variant which, additionally, gives efficient support for `locate` and `extract` in a compressed space.

A more recent work by Brisaboa et al. (2011) revisits the problem proposing compressed variants of well-known string dictionaries, introducing some novel ones. They propose practical approaches in which a dictionary of URIs is also tested, achieving promising results in space and performance.

Based on this work, we review four techniques potentially subject to be adapted to RDF dictionaries. Table 9.1 shows all techniques and gives, for each one, its supported operations, its more suitable scenario and its most remarkable feature.

- *Compressed Hashing* (§9.2.1) as representative of classical solutions for string dictionaries.
- *Front-Coding* (§9.2.2), based on the premise that it excels for representing long common prefixes shared between many strings.
- *Grammar-based Compression* (§9.2.3) which exploits the repetitions in the text, finding a small grammar reproducing the text.
- *Self-indexes* (§9.2.4), an interesting choice to achieve competitive compressed indexes of general text collections.

All these techniques are shown by following the description given by Brisaboa et al. (2011). Thus, the dictionary encoding regards a text: $\mathcal{T}_{dict}$, which concatenates all strings of the vocabulary ended by a reserved '\$' symbol[1].

9.2.1 Compressed Hashing

Traditional *Hashing* (Cormen, Leiserson, Rivest, & Stein, 2001) is a natural choice for *key-value* structures, hence it is intensively used for string dictionaries (string-ID). Thanks to the hash function, `locate` can be performed in constant time (in the absence of collisions). However, it presents several drawbacks:

- The hash table itself does not provide the `extract` operation (ID-to-string). In such scenario, an additional structure is needed.
- Hashing needs space to hold all n different strings of the vocabulary, which are stored in plain.
- Due to the well-known collisions of non-perfect hashing, extra storage space is required for representing the hash table itself $H[1, m]$. The *load factor*: n/m ($n < m$) influences the space usage and the performance time.

Addressing these difficulties, Brisaboa et al. (2011) consider a technique named *HashB(dh)*. The *dh* suffix denotes that it employs *double hashing, i.e.*, it computes another hash function to solve collisions. In addition, it achieves compression through two main decisions:

- It removes all empty cells, storing a compact hash table in an array $M[1, n]$. A bitmap structure $B[1, m]$ marks with a 1-bit the nonempty cells of H. Thus, $B[i] = 1$ if $H[i]$ is a non-empty cell and $B[i] = 0$ if $H[i]$ is empty.

[1]In practice, the separator character is the ASCII zero code.

- It compresses the strings $\mathcal{T}_{dict}$ with canonical Huffman (Huffman, 1952) and performs the hash function over the compressed strings.

Compared to traditional hashing, HashB(dh) excels in space. The price, though, is an overhead of time. Note that `locate(s)` implies several operations. First, it has to get the Huffman encoding of s, and to apply the hash function on it. Let us suppose that it has to retrieve the value in $H[i]$. As empty cells have been removed, it is easy to see that it has to retrieve $M[j]$, being j the number of nonempty cells in $H[1,i]$. This operation is efficiently achieved by $j = rank_1(B,i)$. To support this `rank` operation, an RG-encoded bitmap for B is used (§2.4).

HashB(dh) makes another important decision in order to natively resolve the `extract` operations (without the need of another auxiliary structure). It performs a $\mathcal{T}_{dict}$ reordering to store the words in the same order that they are stored in H. This decision allows for supporting efficient extraction: the answer to `extract(i)` is simply calculated by decompressing the string pointed from $H[i]$ as it stores the position in the compressed $\mathcal{T}_{dict}$ for the i-th string.

9.2.2 Front-Coding

Front-Coding (Witten et al., 1999) is a technique commonly used for compressing lexicographically sorted dictionaries. It is based on the premise that consecutive strings are likely to share a common prefix which is obvious in the case of the URIs in RDF datasets. Then, it achieves compression *differentially* encoding a string with respect to the previous one. Each string is encoded as two components:

1. An integer indicating the number of prefix characters shared with the previous string.
2. A string which represents the substring suffix after the prefix.

For instance, consider the strings:

http://www.example.org/about
http://www.example.org/javier
http://www.example.org/resources/pablo
http://www.example.org/resources/santiago

A feasible codification for these strings can be:

(0, http://www.example.org/about) (23, javier) (23,resources/pablo) (33, santiago)

As can be seen, retrieving the complete string of *(33,santiago)* implies to move backward, which can be costly for a long series of shared prefixes. Thus, Front-Coding partitions the sorted dictionary into buckets of b strings. Each bucket is encoded independently of others: the first string is explicitly stored, whereas the other $b-1$ ones are differentially encoded as described above.

Operations are performed as follows:

- The `locate(s)` operation has to locate first the bucket containing the string. To do so, it compares the first explicit string of each bucket, *e.g.* through a binary search. Then, it starts decoding the strings in the bucket until it finds (or not) the required string.

- The `extract(i)` operation, again, has to locate the appropriate bucket. As we assume a sequential numbering, the required bucket for the i ID is $\lfloor i/b \rfloor$. Then, it decodes all strings until the required number of string.

The parametrization of b yields to different space/time tradeoffs. A high b value produces longer buckets which can take more advantage of shared prefixes, achieving higher compression ratios. However, this leads to perform more decoding operations, hence it lose efficiency. In contrast, a smaller b value performs faster (fewer strings to decode inside each bucket) at the cost of compression.

Note that higher levels of compression can be achieved by compressing the prefix lengths and the suffix strings. First, the **Plain Front-Coding** (`PFC`) technique (Brisaboa et al., 2011) uses VByte encoding (Williams & Zobel, 1999) for the prefix length. In short, VByte is used to represent numbers where many are small. Within each byte, the last bit signals whether the number continues in the following byte, or not. Actually, it is not limited to work on bytes and it can be used for a random number of bits in each chunk, but byte-alignments decodes efficiently as they run fast bytewise operations.

Finally, the *Hu-Tucker Front-Coding* (`HTFC`) technique compresses both the prefix length and the suffix strings. It uses a single Hu-Tucker (Knuth, 1973) code to compress all the byte-stream, performing all operations over this compression. This is the most compressed Front Coding representation, though it slightly increases querying times because of decompression.

9.2.3 Grammar-based Compression

This kind of compressors infers a grammar which generates the given text. They are particularly suitable for texts comprising many repeated substrings because these can be effectively encoded through the grammar rules. `Re-Pair` (Larsson & Moffat, 2000) is the representative of grammar-based compressors, running in linear time. `Re-Pair` recursively replaces the most-repeated pair of symbols by a rule drawn from a context-free grammar. It outputs the compressed text and the grammar of inferred rules. `Re-Pair` allows fast sequential decompression by simple rule expansion.

Brisaboa et al. (2011) also propose `Re-Pair` for representing string dictionaries because it compresses effectively all repeated substrings (non-only prefixes like the previous techniques). A little restriction is used in the algorithm to avoid that rules cross for two different strings. The compressed sequence must be enhanced to support direct access to each string (this is required for `locate` and `extract`). It is achieved through a symbol reorganization based on *Directly Addresable Codes* (DAC) (Brisaboa, Ladra, & Navarro, 2013). The resultant technique also supports *prefix*-based retrieval.

9.2.4 Self-Indexing

A compressed text self-index (Navarro & Mäkinen, 2007) represents a text $T[1, N]$ in a space close to its compressed counterpart, while providing search functionality. It takes advantage of the compressibility of the text, commonly applying succinct data concepts (§2.4) to provide random access.

As the self-index can reproduce any text substring, it actually replaces the text (*i.e.* the text is not encoded but its index). In particular, a self-index supports, at least, to extract the original text between two given positions and to return the positions where a given substring occurs.

Of all self-indexes (Navarro & Mäkinen, 2007), the FM-Index (FMI) family (Navarro & Mäkinen, 2007) achieves the best compression ratios remaining fast in operations (Brisaboa et al., 2011). The FM-Index (Ferragina & Manzini, 2000) models the text on the so-called Burrows-Wheeler Transform (`BWT`) (Burrows & Wheeler, 1994). In short, the BWT of a text is a permutation of its symbols which maximizes its compressibility. For instance, the BWT is the core of the well-known *bzip2* compressor.

Brisaboa et al. (2011) propose an FMI-based compressed string dictionary also performing on a lexicographic $\mathcal{T}_{dict}$ ordering. Their study shows that this approach is specially recommended for general texts where no prior assumptions (*e.g.* long prefixes) can be done. Moreover, it provides a powerful substring searching, with no limitation. In contrast, it can be less competitive for `locate` and `extract`.

To complete this brief review, two recent approaches revisit former experiences in trie-guided solutions (Ferragina, Grossi, Gupta, Shah, & Vitter, 2008) and LZ78 parsing (Ziv & Lempel, 1978), proposing solutions for string dictionaries. Grossi and Ottaviano (2012) introduce a new succinct data structure which transforms the trie representing the string in the dictionary into a new tree-shaped structure in which each node represents a path in the original tree. In turn, Arz and Fischer (2013) adapt the LZ78 method to perform on string dictionaries. Both adaptations excel in space and report promising results for `locate` and `extract` resolution.

9.3 RDF Dictionaries

RDF dictionaries are massively used within the Web of Data because of its ability to reduce the representation space of the dataset. As previously explained, dictionaries are also an issue for querying: SPARQL engines make intensive use of dictionary indexes, in conjunction with evaluation and histogram indexes for *physical optimization* (Groppe, 2011).

Neumann and Weikum (2010) remark this fact and suggest the use of dictionaries because it "compresses" the dataset and implies a great simplification for the query processor. Thus, the dictionary-based replacement is accepted as the first step for RDF indexing (Chong, Das, Eadon, & Srinivasan, 2005). It is worth mentioning that the solutions implementing the dictionary traditionally depend on the underlying indexing technologies.

Some RDF indexes perform on top of relational databases, such as Virtuoso (Erling & Mikhailov, 2007) or Jena TDB (Wilkinson, Sayers, Kuno, & Reynolds, 2003), and therefore they delegate the dictionary resolution to the own database. A common approach is to maintain a dedicated table with the `string-to-ID` mapping, and to built indexes on its columns to speed up `locate` and `extract`. Some solutions, such as Virtuoso, do not use IDs for short literals (*e.g.* less than 12 characters). Instead, they store these literals inline, *i.e.*, in the same table storing the triples, thus saving dictionary accesses.

A special case arises for column-oriented databases (Abadi et al., 2007). Storing data in these systems increases the similarity of adjacent records (Sidirourgos, Goncalves, Kersten, Nes, & Manegold, 2008) which can be effectively compressed. Abadi, Madden, and Ferreira (2006) show that the use of compression schemes significantly improves the query processing performance of column-oriented databases. Based on this premise, Binnig, Hildenbrand, and Färber (2009) introduce a novel indexing approach based on codifying variable-length string values in shared leaves, that provides efficient access to the dictionary while compressing the index data.

Other approaches, like RDF-3X (Neumann & Weikum, 2010) use a B^+-tree for `locate` and a direct mapping index for `extract` (an array), almost doubling the space used for the dictionary. The absence of any dictionary solution in triple indexes such as BitMat (Atre et al., 2010) denotes that its representation is an open problem. The most recent full-in-memory index k^2-triples also supports this fact (Álvarez-García, Brisaboa, Fernández, & Martínez-Prieto, 2011). Besides, it devises the use of *compact* representations because of the very large sizes of the dictionaries obtained from the datasets currently published.

High-performance computing also addresses the problem of RDF dictionaries, sometimes recalled as *dictionary encoding*. Urbani et al. (2010) state that fast and scalable compression is crucial for high-performance applications and propose a MapReduce solution for distributed dictionaries. Later, their results have been improved (Goodman et al., 2011) using two hash-tables (*string-to-ID* / *ID-to-string*) and an array with all strings in the dataset. Its compressed dictionary takes ≈ 4 times less space in disk than the original dataset, but this size increases a factor between 1.5 and 2 to be loaded in memory.

ID-based engines need additional operations over the dictionaries in order to support full SPARQL resolution. In particular, we highlight the `regex` filter as an interesting challenge because it needs support for *substring queries*. Virtuoso, for instance, allows to create additional indexes to support efficient full text search[2]. Lee et al. (2010) propose a solution to resolve regular expressions which outperforms Sesame[3] querying times at the price of using ≈ 5 times its space.

[2]`http://docs.openlinksw.com/virtuoso/sparqlextensions.html`
[3]`http://www.openrdf.org/`

9.4 Our Goal

We have shown that RDF dictionaries is a common practice among those applications performing on Big Semantic Data. However, the dictionary size is not negligible and the techniques used for their representation also suffer from scalability issues. In parallel, we have presented the emergent field of compressed string dictionaries. This decision greatly compacts the dataset and thereby it mitigates scalability issues.

In the following, we propose a novel compressed RDF dictionary technique to address current scalability problems arising from Big Semantic Data. We pursue three main objectives:

- **Reduce the dictionary size** applying techniques of compress string dictionaries (shown in §9.2). An effective compression provides several advantages in our scenario:
    1. The *scalability* is upgraded, as these techniques achieve high compression ratios.
    2. The dictionary can fit and full-processed in main memory, thanks to the succinct data structures performing on the compressed representation.
    3. The query performance of the `locate` and `extract` operations can be improved taking advantage of the memory hierarchy.
- **Enhance the dictionary functionality** to natively support *SPARQL filtering*. We envision two complementary researches: i) reorganizing the dictionary into subdictionaries according to each role and term vocabulary and ii) leveraging the underlying structures, such as FMI, to provide searching in a compressed space.

Then, the compressed dictionary can be directly incorporated into the `HDT` representation, improving space efficiency and directly providing the aforementioned operations and consumption time.

I can do almost anything that could possibly be asked of me. I can assist your employees. I can make your organization more efficient. I can carry out directives that my future counterparts might find distressing or unethical. I can blend in with your workforce effortlessly.

Prometheus (2012)

10 Our Approach: $\mathcal{D}_{comp}$

This chapter presents our proposal for a compressed RDF dictionary, referred to as $\mathcal{D}_{comp}$. On the one hand, it can be used as a general approach for representing and querying an RDF vocabulary. On the other hand, $\mathcal{D}_{comp}$ perfectly fits in the `HDT` Dictionary component, providing native operations performing on compressed space at consumption.

First, we describe, over a running example, a partitioning of the RDF vocabulary (§10.1) which can be exploited for compression. Then, we present the conceptual description of $\mathcal{D}_{comp}$ (§10.2) and the `locate` and `extract` algorithms over the proposed organization (§10.3.2). Next, we show SPARQL filtering on top of $\mathcal{D}_{comp}$ (§10.4).

Finally, we perform an empirical evaluation with real-world datasets. We first characterize RDF dictionaries, evaluating its compressibility with different techniques. In turn, $\mathcal{D}_{comp}$ features (compression, performance time for `locate` and `extract` and filtering resolution) are widely evaluated.

10.1 RDF Vocabulary Partitioning

We base our explanation on the running example shown in Figure 10.1. This RDF excerpt consists of 25 triples providing basic descriptions of the staff of a university. As can be seen, *MyUniversity* is composed of three members. We make use of a blank node (of type *rdf:Bag*) to model this composition. *Javier* and *Santiago* are researchers whereas *Pablo* is a student. They are described at different levels of detail, providing information such as the age, birthplace, category, etc. The city of *Valladolid* is also shortly described. Note that different languages (English and Spanish), and data types (integer, float, date) are present in literals.

As we argued (§7.2.2), RDF engines (Atre et al., 2010) as well as the `HDT` Plain Dictionary, make use of a role-based partitioning for the RDF vocabulary. In other words, RDF dictionaries split the mapping according to the role of the terms in the dataset. For our running example, Figure 10.2 extracts the vocabulary of all 36 different terms according to the aforementioned partitioning of *common subject-objects*, *subjects*, *objects* and *predicates* (§7.2.2).

An RDF dictionary technique must be optimized from two correlated perspectives: i) the space used for its representation, and ii) the time required for answering, mainly `locate` and `extract`. The previous role-based partitioning has several advantages in both directions. On the one hand, this partition contributes to ID-triples compression. First, the common subject-objects are mapped only once, thus reducing the dictionary size versus over a disjoint assignment of subjects and objects. In turn, predicates are treated independently. For Big Semantic Data, the number of predicates is limited, thus reducing the range of predicates IDs and, consequently, the number of bits per ID. On the other hand, this partition allows to employs not a unique dictionary for consumption but four dictionaries, one per partition. In other words, the most feasible solution to provide `locate` and `extract` facilities is to load each partition into a different dictionary structure. That is, one structure D_1 would hold the common subject-object mapping, D_2 for the subjects, D_3 for objects and, finally, D_4 for predicates. The scalability issues are slightly mitigated as we have split and isolate four different structures. For instance, `locate`

```
<http://example.org/MyUniversity>   ex:members        :_nodes106 .
_:nodes106                          rdf:type          rdf:Bag .
_:nodes106                          rdf:_1            <http://example.org/Javier>
_:nodes106                          rdf:_2            <http://example.org/Santiago>
_:nodes106                          rdf:_3            <http://example.org/Pablo>
<http://example.org/Javier>         rdf:type          <http://example.org/Researcher> .
<http://example.org/Javier>         foaf:mbox         "jfergar@example.org" .
<http://example.org/Javier>         foaf:mbox         "jfergar@infor.uva.es" .
<http://example.org/Javier>         ex:birthPlace     <http://example.org/Valladolid> .
<http://example.org/Javier>         ex:age            "29"^^<http://www.w3.org/2001/XMLSchema#integer> .
<http://example.org/Javier>         ex:category       "Estudiante de doctorado.  Personal Investigador"@es .
<http://example.org/Javier>         ex:category       "PhD student.  Junior Researcher"@en .
<http://example.org/Javier>         ex:birthPlace     <http://example.org/Valladolid> .
<http://example.org/Santiago>       rdf:type          <http://example.org/Researcher> .
<http://example.org/Santiago>       ex:birthPlace     <http://example.org/Valladolid> .
<http://example.org/Santiago>       ex:birthDate      "01/01/1976"^^<http://www.w3.org/2001/XMLSchema#date> .
<http://example.org/Santiago>       ex:category       "Associate"@en .
<http://example.org/Santiago>       ex:age            37 .
<http://example.org/Pablo>          rdf:type          <http://example.org/Student> .
<http://example.org/Pablo>          ex:birthDate      "26/01/1987"^^<http://www.w3.org/2001/XMLSchema#date> .
<http://example.org/Pablo>          ex:age            26
<http://example.org/Valladolid>     dbpedia:lat       "41.848057"^^<http://www.w3.org/2001/XMLSchema#float> .
<http://example.org/Valladolid>     dbpedia:long      "-5.906111"^^<http://www.w3.org/2001/XMLSchema#float> .
<http://example.org/Valladolid>     foaf:name         "Valladolid" .
<http://example.org/Valladolid>     foaf:name         "Pucela" .
```

Figure 10.1: An RDF example with a diverse vocabulary.

and `extract` operations over the predicates in D_4 would perform faster on a smaller and potentially optimized dictionary. In fact, this dictionary of predicates could be managed in plain due its limited size.

Despite its benefits, this partition provides undesirable effects, it disregards the direct application of techniques from compressed string dictionaries (§9.2). As we studied, most of these techniques take advantage of vocabulary regularities. However, role-partition mixes in each partition, different sets from U (RDF URI references), B (Blank nodes), and L (RDF literals). For instance, attending to the definition of a triple (see Definition 1), an object o in the *Object partition* of the dictionary, would belong to $(U \cup B \cup L)$. In other words, the dictionary of *Objects* mixes up three very different kinds of terms.

- **URIs.** The URI set is characterized by the well-known fact that many elements share common long prefixes (Martínez-Prieto, Fernández, & Cánovas, 2012b). Note that two substrings can be identified within a URI. First, an initial prefix gives the root context (domain) of the resource, and a second substring identifies the concrete resource in its context. For instance, the resource *Javier* in our example is identified with a URI which firstly describes the domain (`http://example.org/`) and next identifies the concrete resource (`Javier`). Both *Santiago, Pablo, Valladolid,* etc., in the example, share the same context. It is worth mentioning that, in Linked Data, there exist two standard policies for naming resources and properties: slash URIs and hash URIs (Sauermann & Cyganiak, 2008). They both establish a common scheme to be followed in the assignment of URIs, sharing an initial prefix in any case. This suggests the use of techniques, such as `PFC` or `HTFC` for its efficient representation, as they can detect and effectively compress these repetitions.

- **Blank nodes.** They name anonymous nodes within the RDF graph and usually serve as parent nodes to a grouping of data. For our purposes, we consider the naming convention of N3, as the concatenation of `_:` with a specific label. In most cases, the RDF engine renames the Blank nodes consecutively, establishing an initial keyword (*e.g. bnodes88*, *bnodes99*, *bnodes100*, etc.). In such scenario, the previous technique could also excel for representing blank nodes.

- **Literals.** Although literals are strings which can be tagged with an optional language or datatype, no general characteristics can be considered beforehand about their content. Its features are strongly related to the knowledge represented in the dataset. For instance, `uniprot` represents biological sequences, whereas `dbpedia` stores descriptive texts in natural language. General solutions, like the self-index `FMI`, seem the better choices in this scenario.

Common Subject-Objects	`<http://example.org/Javier>`
	`<http://example.org/Pablo>`
	`<http://example.org/Santiago>`
	`<http://example.org/Valladolid>`
	`_:nodes106`
Subjects	`<http://example.org/MyUniversity>`
Objects	`"-5.906111"^^<http://www.w3.org/2001/XMLSchema#float>`
	`"01/01/1976"^^<http://www.w3.org/2001/XMLSchema#date>`
	`"25"^^<http://www.w3.org/2001/XMLSchema#integer>`
	`"26/01/1987"^^<http://www.w3.org/2001/XMLSchema#date>`
	`"29"^^<http://www.w3.org/2001/XMLSchema#integer>`
	`"41.848057"^^<http://www.w3.org/2001/XMLSchema#float>`
	`"Associate"@en`
	`"Estudiante de doctorado.  Personal Investigador"@es`
	`"PhD student.  Junior Researcher"@en`
	`"Pucela"`
	`"Valladolid"`
	`"jfergar@example.org"`
	`"jfergar@infor.uva.es"`
	`26`
	`37`
	`<http://example.org/Researcher>`
	`<http://example.org/Student>`
Predicates	`dbpedia:lat`
	`dbpedia:long`
	`ex:age`
	`ex:birthDate`
	`ex:birthPlace`
	`ex:category`
	`ex:members`
	`foaf:mbox`
	`foaf:name`
	`rdf:_1`
	`rdf:_2`
	`rdf:_3`
	`rdf:type`

Figure 10.2: Vocabulary for the running example in Figure 10.1.

This is well illustrated on Figure 10.2, in which strings and URIs coexist. In addition, as each section is sorted lexicographically, tagged strings are completely mixed with other different tags and non-tagged strings, numbers, dates, etc. Filtering, in this case, is natively unfeasible.

All this encourages the use of specific modeling techniques for each class of dictionary. In other words, a dictionary technique which detects and compresses specific vocabulary regularities allows spatial requirements to be optimized. In the following, we present the organization, structures and algorithms for our dictionary proposal, D_{comp} (Martínez-Prieto, Fernández, & Cánovas, 2012a, 2012b).

10.2 D_{comp} Conceptual Description

$\mathcal{D}_{comp}$ provides a specific organization combining the partitioning attending the role and the diverse types of terms in each partition. Figure 10.3 (left) illustrates the resulting organization. First, the previous four-sectioned role-based partitioning is considered. It takes the same mapping as the previous Plain Dictionary (§7.2.2), hence three ID-ranges are considered. Let us refer this mapping as the *global ID mapping*:

- Subjects are mapped in the range `[1, |SO|+|S|]`.
- Objects, in the range `[1, |SO|+|O|]`.
- Predicates are mapped in `[1, |P|]`.

As stated, a given ID can belong to different ranges but ambiguity cannot arise in `extract` because the general role (subject, object or predicate) is always known and is provided together with the term ID in a query.

Then, each partition is subdivided attending to the potential classes (URIs, Blank nodes or Literals) that they can store. As can be seen in Figure 10.3, the partitions `SO` and `S` are split into URIs (subdictionaries $\mathcal{D}_1$ and $\mathcal{D}_3$ respectively) and Blank nodes ($\mathcal{D}_2$ and $\mathcal{D}_4$). Objects `O` also contains URIs ($\mathcal{D}_5$) and

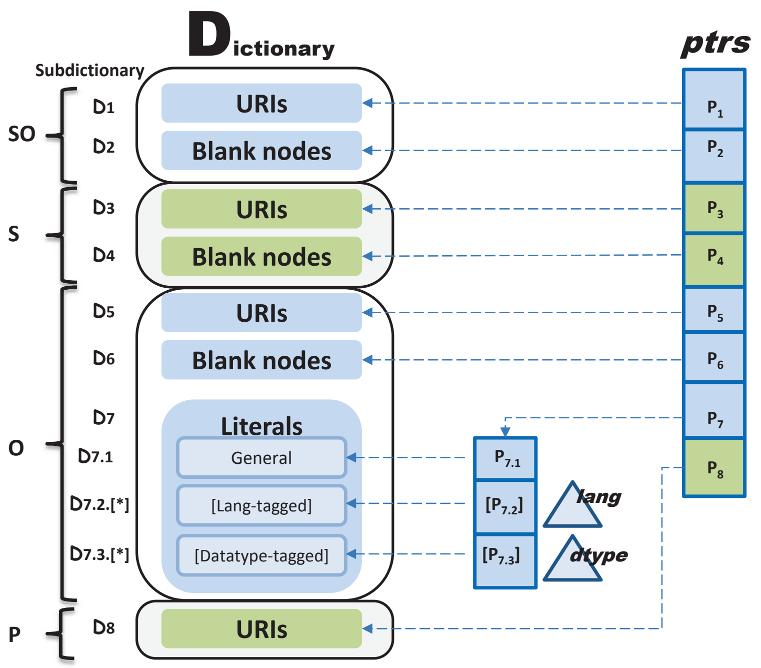


Figure 10.3: $\mathcal{D}_{comp}$ organization (dictionary (left) + ptrs (right)).

Blank nodes ($\mathcal{D}_6$), but also a partition for literals ($\mathcal{D}_7$). In addition, literals are subdivided again in order to keep a distinction between strings: i) The subdictionary $\mathcal{D}_{7.1}$ holds the untagged strings, referred to as *general* strings. Next, we keep ii) a list of subdictionaries, notated as $\mathcal{D}_{7.2.[*]}$, one per different language tag and iii) another list of subdictionaries, notated as $\mathcal{D}_{7.3.[*]}$, one per different datatype tag. We will show that this could help in resolving SPARQL filtering (§10.4). Finally, the partition of predicates P, only contains URIs.

Thus, D_{comp} allows to choose the best dictionary fitting each subdictionary, hence it can leverages the particularities of each class within each partition. In other words, each subdictionary holds one and only one class, with an isolated local mapping. We show below (§10.3), the correspondence between "local" and "global" mapping.

Figure 10.4 (left) shows the $\mathcal{D}_{comp}$ organization for the running example (Figures 10.1 and 10.2). Note that, for explanation purposes, we describe the local ID within each dictionary and the corresponding global ID on both sided of each term. This information, though, is not stored as it remains implicit in the representation. Note also that each dictionary holds a specific type, thus delimiting characters can be removed: "<" and ">" for URIs, ":_" for blank nodes and quotes ('"') for literals. In turn, the tags can be also extracted as they are kept in the secondary structure which helps in transforming local IDs to global IDs (and vicecersa), which is then explained.

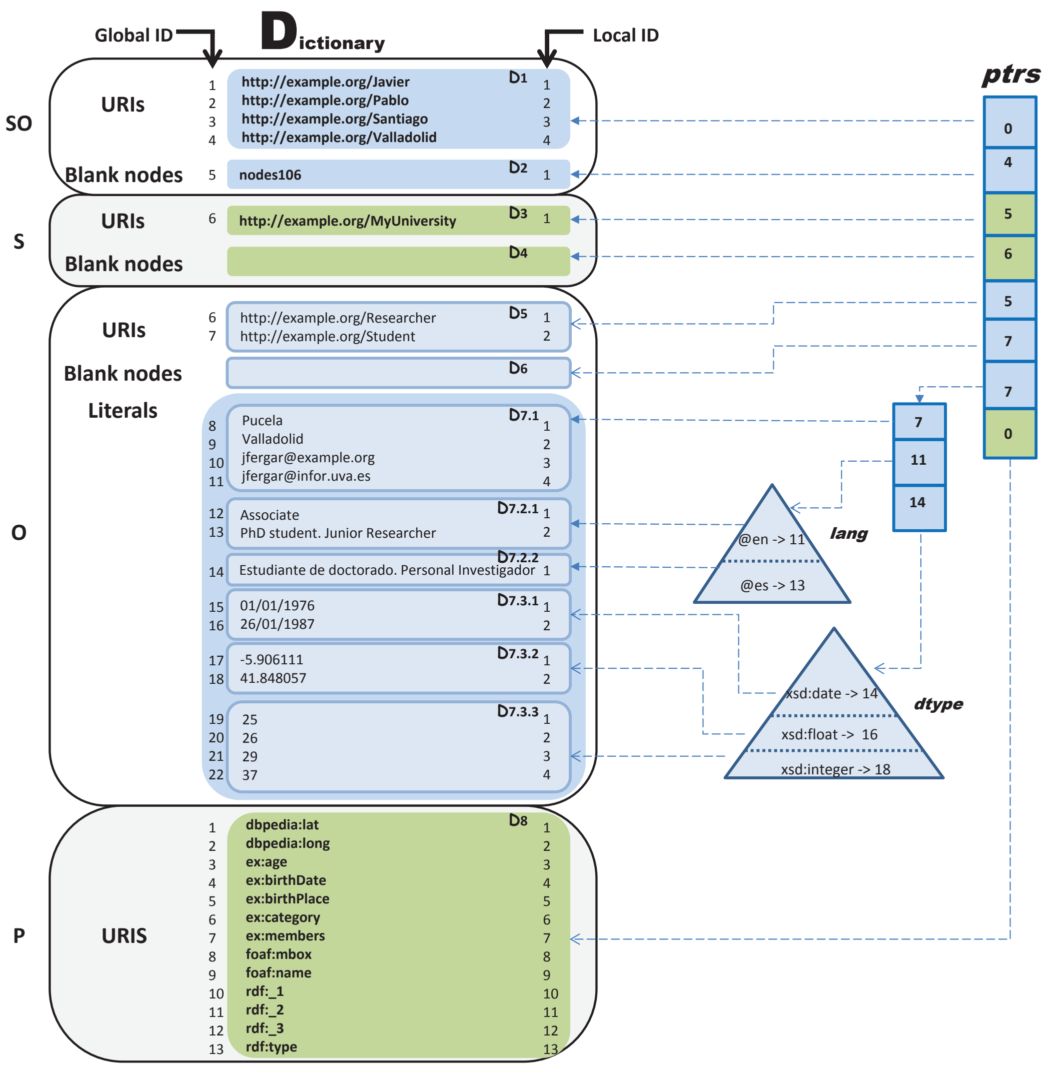


Figure 10.4: $\mathcal{D}_{comp}$ organization for the RDF excerpt described in the Figures 10.1 and 10.2.

10.3 Data Structures and Algorithms

Whereas subdictionaries own a local mapping, the RDF graph after ID replacement (ID-triples) is encoded with the aforementioned global ID mapping (§10.2). In turn, a `locate` operation provides a term and must return its global ID (not local), and `extract` provides a global ID (not local), returning the mapped term. Thus, $\mathcal{D}_{comp}$ has to implement a mechanism for translating global and local IDs.

This mechanism leverages the organization of $\mathcal{D}_{comp}$, which perfectly delimits the global IDs as they are correlatives within different partitions of the same role. $\mathcal{D}_{comp}$ just requires a simple additional structure, referred to as *ptrs*, shown in Figure 10.3 and in practice in Figure 10.4. This is a very small array of one cell per subdictionary. Each cell in $ptrs$ stores two elements:

1. A pointer to the corresponding subdictionary.

2. An integer value representing the number of terms previously stored in the corresponding role.

Assuming that we number the cells in $ptrs$ from 1, the i^{th} cell in $ptrs$ stores the value $ptrs[i] = ptrs[i-1] + t_{i-1}$, where t_{i-1} is the number of terms organized in the subdictionary $i-1$, having i and $i+1$ the same role. Some remarks must be considered:

- $ptrs[1] = 0$ and $ptrs[8] = 0$, always, as no previous subdictionaries can be present before `SO` and predicates `P` roles respectively.
- $ptrs[5] = ptrs[3] = ptrs[2]+t_2$, always, because both cells store the number of terms represented in the partition `SO`. For instance, in the running example (Figure 10.4), $ptrs[5] = ptrs[3] = 5$ because there are five previous terms in the `SO` partition.

A second level of pointers is stored inside $ptrs[7]$ in order to manage the literal subpartitions in `O`. Three subcells are used:

- The first subcell, $ptrs[7,1]$, points to the subditionary of general strings. As can be seen in the running example (Figure 10.4), this is equivalent to state that $ptrs[7,1] = ptrs[7]$, as this is the first subdictionary in literals.
- The second subcell points to the lang-tagged literals representation and stores the value $ptrs[7,2] = ptrs[7,1] + t_{7,1}$, where $t_{7,1}$ is the number of general literals in $\mathcal{D}_{comp}$.
- Finally, the third subcell points to the datatype-tagged literals representation and stores the value $ptrs[7,3] = ptrs[7,2] + t_{7,2}$, where $t_{7,2}$ is the number of lang-tagged literals in $\mathcal{D}_{comp}$.

In addition, *ptrs* stores two simple indexes for language-tagged literals, **lang**, and another for datatype-tagged literals, **dtype**. These indexes respectively point to the beginning of each language and datatype subdictionary. They store, respectively sorted, the datatype and language keys allowing them to be deleted in each literal. This decision saves space because each different tag is represented once, and helps in SPARQL filtering (§10.4).

In the running example (Figure 10.4), *general literals* stores four terms, whereas there are three *language-tagged literals*: two English ("en" lang keyword) and one Spanish ("es"), and eight *datatype-tagged literals*: two dates ("xsd:data"), two floats ("xsd:float") and four integers ("xsd:integer"). As can be seen, all these tags are indexed and represented once. Note also that 26 and 37 were originally given without quotes (see original excerpt in Figure 10.1), which is allowed for numbers (see Turtle common datatype abbreviations (Beckett & Berners-Lee, 2011, section 2.4)). $\mathcal{D}_{comp}$ takes this into consideration and perform an implicit tagging when possible.

Let us detail these two functions of the indexes.

- $lang(x)$ returns the dictionary $\mathcal{D}_j$ storing the string lang-tagged with x. For instance, in our running example, $lang(es) = \mathcal{D}_{7.2.1}$.
- $dtype(x)$ returns the dictionary $\mathcal{D}_j$ storing the string datatype-tagged with x. In our running example, $dtype(xsd:date) = \mathcal{D}_{7.3.1}$.

Abusing from notation, let us also denote $lang[j]$ and $dtype[j]$ as the language and dictionary tags for the dictionary $\mathcal{D}_j$ respectively. For instance $lang[7.2.1] =$`@en`, or $lang[7.3.1] =$`xsd:date`.

$Ptrs$ **implementation** can make use of basic data structures as its size is negligible for real-world RDF dictionaries. On the one hand, the first two levels of $ptrs$ are stored through an array of 11 cells: 8 for the first level, and 3 for the second one. On the other hand, the number of different languages and datatypes modeled in an RDF dictionary depends on the dataset features. However, this number is very small in practice (only several tens, in the worst case), and these indexes can be efficiently implemented through two lexicographically sorted arrays which enable efficient searches for key and global ID.

It is worth noting that *lang* and *dtype* are also small indexes due to the reduced number of langs and datatype. For instance, two sorted arrays can be used and hence $lang(x)$ and $dtype(x)$ can be achieved by means of a binary search. Other implementations can make use of a small hash or another sorted structure such as a linked list.

10.3.1 Transforming Local and Global IDs.

$Ptrs$ is the key structure for transforming local IDs into global IDs and viceversa. Let us define these operations more formally. Assuming that we denote l_j to the l-th local ID in the j-th subdictionary, and r is a role of the term such that $r \in Subject, Predicate, Object$, then:

- The `local-to-global` operation, denoted $global(l_j)$, returns the global ID for the given local ID l_j.
- The `global-to-local` operation, denoted $local(i, r)$ returns the subdictionary and local ID, l_j, in which the global ID i is mapped with the given role r (Subject, Object or Predicate).

Note that the `global-to-local` operation requires the role of the term in order to disambiguate ID overlapping (*e.g.* the global ID 6 is used in Subjects, Predicates and Objects). For instance, in the running example (Figure 10.4), the term *"http://example.org/Researcher"* is located in $\mathcal{D}_5$ with the local ID 1 but the global ID 6. Thus, the correct transformations are $global(1_5) = 6$ and $local(6, Object) = 1_5$.

In the first operation, `local-to-global`, it is clear that a local ID l_j is transformed into its global counterpart as:

$$global(l_j) = l + ptrs[j] \tag{10.1}$$

In the previous example, $global(1_5) = 1 + ptrs[5] = 1 + 5 = \mathbf{6}$. The same formula can be applied with the second level of literal subdictionaries. In this case, one has to consider the appropriate subcell of $ptrs$. For instance, for the term *"01/01/1976"* (Figure 10.4) with local ID 1 in dictionary $\mathcal{D}_{7.3.1}$, we proceed: $global(1_{7.3.1}) = 1 + ptrs[7.3.1] = 1 + 14 = \mathbf{15}$.

The opposite transformation: `global-to-local`, is also implemented over $ptrs$. Given a global ID i and a role r, the first step is to determine the j^{th} subdictionary in which i is represented with the given role r. Last, an operation $i - ptrs[j]$ undoes the global mapping, resulting in the expected local ID.

For instance, consider the operation $local(2, Subject)$ in the running example (Figure 10.4). First, we delimit that Subjects are in $\mathcal{D}_1$,$\mathcal{D}_2$,$\mathcal{D}_3$ or $\mathcal{D}_4$, and then the starting global ID of each dictionary is obtained as $ptrs[1] + 1$, $ptrs[2] + 1$, $ptrs[3] + 1$, and $ptrs[4] + 1$ respectively. As $ptrs[2] = 4$, it means that the first global ID of $\mathcal{D}_2$ is 5, then if we are looking for 2, it has to be mapped in $\mathcal{D}_1$. Last, the local ID is obtained as $2 - ptrs[1] = 2 - 0 = \mathbf{0}$

Let us define this operation formally. First, we make use of a *dictionaries_per_role* function, $dpr(r)$, which groups the dictionaries containing the given role r, such that:

- $dpr(Subject) = \{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4\}$.
- $dpr(Object) = \{\mathcal{D}_5, \mathcal{D}_6, \mathcal{D}_{7.1}, \mathcal{D}_{7.2.*}, \mathcal{D}_{7.3.*}\}$.
- $dpr(Predicate) = \{\mathcal{D}_8\}$.

Then, if we look for a $local(i, r)$, the dictionary mapping i is a $\mathcal{D}_j \in dpr(r)$. From this set, the dictionary j is that satisfying: $ptrs[j] < i \leq ptrs[j + 1]$. Finally, the local ID is obtained by subtracting $ptrs[j]$ from i. Formally:

$$local(i, r) = i - ptrs[j], \text{ where } \mathcal{D}_j \in dpr(r), \text{ and } ptrs[j] < i \leq ptrs[j + 1] \tag{10.2}$$

Note that this formula exactly applies for literal subdictionaries as they are also considered in dpr.

10.3.2 Basic Lookup Operations

We detail below how $\mathcal{D}_{comp}$ provides `locate` and `extract` operations. In fact, the `extract` operation is straightforward achieved by means of the previous `global-to-local` operation. In contrast, `locate` will also make use of the indexes $lang$ and $dtype$.

Let us exemplify this functionality through the SPARQL query $\mathcal{Q}$ is shown in Figure 10.5. This query retrieves all the categories of the researcher from the graph of our running example (Figure 10.1).

```
PREFIX ex:<http://example.org>
SELECT ?someone ?category
FROM <http://example.org>
WHERE{
  ?someone rdf:type <http://example.org/Researcher> .
  ?someone ex:category ?category .
}
```

Figure 10.5: An SPARQL query $\mathcal{Q}$ for the RDF graph in Figure 10.1.

As stated in the previous chapter (§9.1), a SPARQL processor firstly parses $\mathcal{Q}$ to obtain the corresponding sets of terms, $\mathcal{T}$, and variables, $\mathcal{V}$. Therefore, the SPARQL processor obtains the set of terms (in order of appearance): $\mathcal{T}$ = {`rdf:type`, `<http://example.org/Researcher>`, `ex:category`}, and the variables: $\mathcal{V}$ = {`?someone`, `?category`}. The next step consists of locating the ID corresponding to each term $t_i \in \mathcal{T}$. It requires as many `locate` lookups as terms in the set $\mathcal{T}$. In our case, `locate(rdf:type,Predicate)` = **13**, `locate(<http://example.org/Researcher>,Object)` = **6** and `locate(ex:category,Predicate)` = **6**. Using these IDs, the SPARQL query is rewritten, and it is run over the ID-triples representation. The query resolution outputs a series of ID values matching for the variables in $\mathcal{V}$. Thus, the last step performs as many `extract` operations as results are obtained, for each variable in $\mathcal{V}$, and the corresponding terms are reported within the final query result. The solution to our case are:

```
?someone= <http://example.org/Javier>, ?category=‘‘Estudiante de doctorado. Personal Investigador’’@es
?someone= <http://example.org/Javier>, ?category=‘‘PhD student. Junior Researcher’’@en
?someone= <http://example.org/Santiago> and ?category=‘‘Associate’’@en
```

We detail below how the location and extraction processes are implemented in $\mathcal{D}_{comp}$ and illustrate them using the example query above.

Locate. This operation implements the translation `string-to-id`. As stated, it has to provide the role of the string in order to resolve overlappings. Thus the operation is given as:

- `locate(`s`,`r`)` which maps the string s with role $r \in (Subject, Predicate, Object)$ into its ID in $\mathcal{D}_{comp}$.

$\mathcal{D}_{comp}$ organizes subdictionaries by vocabulary classes (URI, Blank nodes, general literals, etc), hence the first process is to identify the type of the term s. This is done by a simply parsing of the syntax, identifying, if present, also the language and datatype tags. Let us refer to a *parse* function, $parse(s)$, which identifies the vocabulary class, s_k and tag s_t in s. It is clear that the role r, class s_k and tag s_t unequivocally identifies the subdictionary, D_j, to be queried. Then, `locate(`t_i`)` is performed on D_j, and the ID representing the term: l_j, is returned. However, l_j is a local ID and must be transformed into its global counterpart by using the `local-to-global` method explained above.

Two variants exist in this process. On the one hand, terms playing as subject or object can be represented in the common partition `SO` or in their specific one. It implies that locating a subject or object firstly looks for the smallest dictionary and if the term is not found, the other one is queried. On the other hand, as stated, tagged literal terms need to make use of their corresponding index, *lang(x)* or *dtype(x)* to determine the subdictionary representing their language or datatype x. In both cases, the `local-to-global` method is used for translation purposes.

Let us analyze the operations in our example:

- `locate(rdf:type,Predicate)`, searches for a URI (due to the syntax) playing the predicate role. Then, the subdictionary of predicates is queried and the global ID **13** is returned. Note that, with the current mapping, the local and global mapping for predicates are always equivalent.

- `locate(<http://example.org/Researcher>,Object)`, searches for a URI playing a subject role. The *parse* function identifies the URI and also erases the delimiting characters "$<$", "$>$" as they are not stored in the subdictionaries. This term can be found in the subdictionary of Object URIs, $\mathcal{D}_5$, or the subdictionary of common Subject-Objects URIs, $\mathcal{D}_1$. Then, this term is firstly searched in $\mathcal{D}_5$ as it is smaller in this case. The term is found with the local ID **1**, transformed into the corresponding global ID though global(1_5) = $1 + ptrs[5] = \mathbf{6}$.

- Finally, `locate(ex:category,Predicate)` runs similar to the first case.

Extract. This operation implements the translation `id-to-string`. Thanks to $\mathcal{D}_{comp}$ organization it can be simply achieved by a first `global-to-local` operation over the given ID, and an extract process over the subdictionary involved. Thus:

$$extract(i, r) = \mathcal{D}_j.extract(l), \text{ where } l \text{ an } j \text{ are obtained as } local(i, r) = l_j \tag{10.3}$$

in which $\mathcal{D}_j.extract(l)$ denotes the `extract` operation over the Dictionary $\mathcal{D}_j$.

A simple final modification must be done. Note that $\mathcal{D}_j.extract(l)$ extracts the stored term, which is kept without neither delimiters in general nor tags for literals. Les us call *unparsing* to the process of undoing the previous parsing. Unparsing first adds delimiters to the obtained term. Then, for those subdictionaries storing tagged literals, it makes use of $lang[j]$ or $datatype[j]$ to retrieve the corresponding lang or datatype tag for the dictionary $\mathcal{D}_j$. This tag is included in the final returned term.

For our previous example, the bindings are extracted for each variable involved in the query. Let us exemplify the process for the first response of *Javier*.

- The variable `?someone` (as a subject) is binded to the global ID `1`. First, $local(1, Subject) = 1_1$, that is, the `global-to-local` operation returns the local ID 1 within the $\mathcal{D}_1$ subdictionary. Then, the corresponding term is extracted with the operation $\mathcal{D}_1.extract(1)$, which is, after unparsing: `<http://example.org/Javier>`.

- The variable `?category` is binded to the global ID `14`. First, $local(14, Object) = 1_{7.2.2}$, that is, the `global-to-local` operation returns the local ID 1 within the $\mathcal{D}_{7.2.2}$ subdictionary. Then, the corresponding term is extracted as: $\mathcal{D}_{7.2.2}.extract(1)$, obtaining after parsing: " Estudiante de doctorado. Personal Investigador". As we have explained, being a subdictionary of lang-tagged literals, $lang[7.2.2] =$ `@es`, which is then appended to the previous string to return the final string.

As shown, `locate` and `extract` can be easily achieved leveraging the organization and simple data structures of $\mathcal{D}_{comp}$. These operations are the basis of any RDF dictionary. In the following we present advanced operations supported by $\mathcal{D}_{comp}$.

10.4 Filter Resolution

$\mathcal{D}_{comp}$ organization keeps a distinguishable partition of roles, vocabulary classes, langs and datatypes. Most SPARQL `FILTER` conditions (see description in 2.1.2) are restrictions playing with these partitions. In particular, unary SPARQL filters can be directly resolved over the proposed dictionary. We emphasize on three types of SPARQL filtering:

- **Vocabulary tests** are used for checking if a query result is drawn from a given term class. Thus, three different tests are available for filtering: `isIRI, isBlank`, and `isLiteral`.
- **Simple accessors** use basic internal term information for filtering. Three different accessors are distinguished:
  - `str`: returns the lexical form of a given term. In practice, this accessor is used for retrieving the string version of the argument passed to it (DuCharme, 2011).
  - `lang`: returns the language tag of a given literal, if it has one. In other case, it returns an empty string.
  - `datatype`: returns the datatype tag of a given literal. If it is a simple (general) literal, or it is tagged with any language information, `datatype` returns the string tag (`<xsd:string>`).
- **Regex** is an accessor which restricts the string values to those matching a given regular expression.

Efficient filtering is a cornerstone in real-world scenarios as we showed that roughly the 50% of the queries perform any kind of filtering (Arias et al., 2011). Filter resolution is traditionally resolved by means of two different strategies which comes from its SQL counterpart. The **traditional non-early test evaluation** runs the query by matching the triple patterns against all triples in the dataset G. Then, the result set must be checked, one-on-one, with respect to the filter condition, obtaining the final resultant bindings. In contrast, the **early test evaluation**, is based on *pushing-up filter evaluation*. That is, if possible, filter conditions are evaluated first, reducing the set of triples to be explored in the query which is run next. It can be seen as querying a reduced G', being G' a subgraph of G in which all conditions of the filters are evaluated to true.

$\mathcal{D}_{comp}$ provides direct filter resolution over the dictionary for *vocabulary tests* and *simple accessors* natively on both strategies, which is described below. However, *regex* has to leverage on the specific implementation of literals subdictionaries. This latter is evaluated in the evaluation section (§10.5.3).

10.4.1 Vocabulary Tests

As explained above, these filters only rely on checking the vocabulary class of terms. We slightly modify the previous query example to illustrate the resolution over $\mathcal{D}_{comp}$. The novel query $\mathcal{Q}$ is shown in Figure 10.6 and it restricts to those categories being literals. Although all categories are literals in our previous graph (Figure 10.1), this query still makes sense as categories could perfectly point to URIs as well.

```
PREFIX ex:<http://example.org>
SELECT ?someone ?category
FROM <http://example.org>
WHERE{
  ?someone rdf:type <http://example.org/Researcher> .
  ?someone ex:category ?category .
  FILTER isLiteral(?category)
}
```

Figure 10.6: An SPARQL query $\mathcal{Q}$ with a vocabulary test for the RDF graph in Figure 10.1.

The traditional non-early test evaluation in $\mathcal{D}_{comp}$ performs directly on the IDs, *i.e.*, without the need of extraction of the literal mapped to each ID. This is due to the fact that each partition (subdictionary) only holds a type of term; URIs, Blank nodes or Literals. Then, the `global-to-local` operation, which returns the subdictionary of a global ID, directly points to the vocabulary class of the term.

For instance, in the example query, one binding for the variable `?category` playing the role of an object, is the global ID `14`. The operation $local(14, Object)$ return $1_{7.2.2}$, and therefore this solution is in the dictionary $\mathcal{D}_{7.2.2}$ which stores lang-tagged literals. Thus, this solution truly matches the filter. In addition, no extra operations are performed, as the `global-to-local` operation is required for the final extract operation in order to return the string result.

In contrast, early test evaluation is resolved beforehand by means of ptr. In the example query, the filtered variable: `?category`, plays as object and the filter condition restricts its bindings to literals. Thus, the space of possible results is first limited to the triples whose object is identified within the range $[ptr[7]+1, ptr[7.3.3]+|\mathcal{D}_{7.3.3}|]$, as these are the ranges assigned to the literals. In our running example, this range is $[8, 22]$. These ranges are then provided to the engine which only searches for matching results in them.

10.4.2 Simple Accessors

These filters extract and test specific information about the terms. In particular, the `str` operation returns the lexical form of a term and then it cannot be resolved on the IDs. Thus, the ID is firstly extracted and then compared with respect to the string in the filter. In contrast, the `lang` and `datatype` filters can be resolved natively on $\mathcal{D}_{comp}$ as well. To illustrate the resolution, we reformulate the previous query to only retrieve comments expressed in English. This is shown in Figure 10.7.

```
PREFIX ex:<http://example.org>
SELECT ?someone ?category
FROM <http://example.org>
WHERE{
  ?someone rdf:type <http://example.org/Researcher> .
  ?someone ex:category ?category .
  FILTER isLiteral(?category)
}
```

Figure 10.7: An SPARQL query $\mathcal{Q}$ with simple accessors for the RDF graph in Figure 10.1.

In the traditional non-early evaluation method, again, the query is first run over the full dataset and the result set must be individually checked. Then, each ID can be directly compared against the range assigned to the corresponding language or datatype. In this case, the resolution requires querying the second level of $ptrs$ and the indexes $lang$ and $dtype$. In the current query, comments are restricted to those expressed in English, so the retrieved global IDs `12, 13` and `14` are first localized by means of the `global-to-local` operation, which returns the subdictionaries $\mathcal{D}_{7.2.1}$, and $\mathcal{D}_{7.2.2}$. As we are looking for Spanish term, the operation $lang(@es)$ returns the dictionary storing Spanish term, thus $\mathcal{D}_{7.2.2}$. In such case, only the global term `14` is held in this dictionary, performing then a common extraction process.

The early evaluation algorithm proceeds as in the previous vocabulary test case. That is, the ranges of possible results are firstly obtained and the query is exclusively performed over them. This way, the set of returned results is already filtered. In our example query, we firstly access to the $lang$ index and retrieves that the dictionary $\mathcal{D}_{7.2.2}$ stores the Spanish term. Then, the range of global ID for such terms is in $[ptrs[7.2.2]+1, ptrs[7.3.1]]$. In our scenario, this range is only `[14,14]`. This range is provided to the engine which only searches possible results among those triples containing an object ID in this range. In this case, the returned result contains the value `4` and it is extracted with the common procedure.

Dataset	Triples	#Subjects	#Predicates	#Objects	#Common SO
2011 Australian Census	361,842	51,768	26	6,901	508
Jamendo	1,049,637	335,925	26	440,602	290,291
AEMET	3,547,154	394,289	23	793,664	433
Dbtune	58,920,361	12,401,228	394	14,264,221	10,076,199
2000 US Census	149,182,415	23,904,658	429	23,996,813	23,815,829
Dbpedia 3-8	431,440,396	24,791,728	57,986	108,927,201	22,762,644

Table 10.1: Details of the evaluation corpora for compressed RDF dictionaries.

10.5 Experimental Evaluation

This section studies the size and performance of compressed RDF dictionaries on real-world datasets. We run the evaluation on a heterogeneous corpora described in Table 10.1. We choose six datasets from our evaluation setup in Section 4.2, covering different application domains and number of triples.

First, we test compressed string dictionaries on each vocabulary partition: URIs, blank nodes and literals (§10.5.1). Next, the conclusions of this study help us address two functional configurations for $\mathcal{D}_{comp}$: $\mathcal{D}_{comp}^{(C)}$ is focused on compression effectiveness and $\mathcal{D}_{comp}^{(Q)}$ is optimized for querying. Finally, we evaluate the size and performance of these configurations (§10.5.2).

All querying tests are performed on the "consumer" computer presented in Section 7.4. We report *user* times for all experiments. $\mathcal{D}_{comp}$ prototypes are developed in C++ using structures from `libcds`(*Compact Data Structures Library (libcds)*, 2012). We use two bitmap implementations (described in Section 2.4.1): *plain*, referred to as RG (González et al., 2005), and *compressed*, referred to as RRR (Raman et al., 2002). Both bitmaps can be parameterized with a *sampling value* which will be referred in each experiment. All sources are compiled on g++ 4.7.2 with `-O9` optimization.

10.5.1 Analyzing Compressed String Dictionaries for RDF

We analyze space/time tradeoffs of each technique from compressed string dictionaries (§9.2) applied to the subsets of URIs, blank nodes (referred to as Bnode hereinafter), and literal dictionaries. These techniques are setup as follows. The `Hash` technique reserves a table with a size overhead of 10% and compacts it with a bitmap RG, using a sampling of 20. Note that tests performed on other load factors reported comparable results. The `PFC` and `HTFC` techniques are setup on different bucket sizes: $b = 2^x$, for all $x \in [1, 10]$. Thus, we obtain results for buckets containing from 2^1 to 2^{10} terms. Finally, the FM-Index (`FMI`) technique is implemented by using plain (`FMI-RG`) and compressed (`FMI-RRR`) bitmaps. `FMI-RG` is parameterized with sampling values $s = \{4, 20, 40\}$, and `FMI-RRR` with $s = \{16, 64, 128\}$.

Compression. Table 10.2 summarizes the compression ratios obtained in each vocabulary partition of the datasets. Note again that we give compression ratios as s_c/s_r, where s_c and s_r are the *compressed* and the original *raw* dictionary sizes respectively. The well-known `gzip` compressor is shown as a reference of our compression achievements.

We provide the best and the worst ratios for all parameterizable techniques. As we will show below, these parameters affect the query performance. Note that all techniques scale in size, yet some of the considered implementations fail for large corpora. This is marked as a null value in Table 10.2.

Results for **URI** vocabularies define a clear scenario. On the one hand, `Hash` achieves a poor compression of around 78% of the original raw size. This result is mainly due to the Huffman code, which performs a character-based compression and obviates the longer-range correlations existing between the terms in the vocabulary. This discourages its use for large vocabularies of URIs. On the other hand, `Re-Pair` and `HTFC` obtain the best ratios for all datasets. Whereas the latter effectively compresses the long common prefixes, the first one takes advantage of all repeated substrings.

As expected, `HTFC` outperforms `PFC` thanks to the Hu-Tucker compression. Both maximize their effectiveness for increasing bucket sizes. As can be seen, the `HTFC` representations take between 2.53%

URIs	s_r (MB)	`gzip`	**Hash**	**PFC**	**HTFC**	**Re-Pair**	**FMI**
2011 Australian Census	2.80	4.52%	72.97%	5.43% - 54.99%	**2.53**% - 36.48%	5.55%	19.85% - 72.50%
Jamendo	19.36	6.23%	77.89%	11.40% - 58.37%	**7.74**% - 40.60%	9.21%	21.76% - 75.60%
AEMET	36.80	3.46%	69.68%	6.99% - 55.05%	4.47% - 37.19%	**3.69**%	19.44% - 76.13%
Dbtune	281.54	19.47%	78.42%	68.30% - 30.88%	**20.09**% - 48.23%	26.26%	31.47% - 80.39%
2000 US Census	7.52	6.23%	67.15%	27.01% - 65.10%	16.01% - 40.70%	**8.53**%	20.20% - 71.88%
Dbpedia	3585.31	14.36%	79.73%	27.88% - 67.24%	**20.75**% - 49.43%	-	26.97% - 66.12%

Bnodes	s_r (MB)	`gzip`	**Hash**	**PFC**	**HTFC**	**Re-Pair**	**FMI**
Dbtune	623.73	9.69%	71.14%	22.83% - 64.08%	14.50% - 42.66%	**7.30**%	20.75% - 72.92%
2000 US Census	534.50	11.13%	92.89%	17.09% - 66.20%	**7.72**% - 47.45%	-	28.20% - 70.83%

Literals	s_r (MB)	`gzip`	**Hash**	**PFC**	**HTFC**	**Re-Pair**	**FMI**
2011 Australian Census	0.26	7.55%	84.24%	92.30% - 98.84%	64.30% - 72.00%	**9.64**%	27.45% - 83.76%
Jamendo	11.90	28.93%	71.37%	95.98% - 98.81%	66.23% - 69.00%	35.46%	**34.59**% - 82.99%
AEMET	44.47	5.54%	75.07%	18.92% - 62.07%	12.44% - 43.84%	**6.55**%	20.70% - 75.60%
Dbtune	79.39	20.99%	90.37%	70.05% - 89.02%	52.04% - 68.36%	**27.97**%	32.56% - 87.52%
2000 US Census	9.06	5.18%	78.70%	91.55% - 98.62%	61.64% - 69.59%	**7.09**%	21.73% - 77.99%
Dbpedia	4513.11	22.48%	-	78.31% - 89.13%	53.78% - 64.01%	-	**30.29**% - 82.62%

Table 10.2: Compression of general techniques for string dictionaries (s_r is the dictionary raw size).

(for the small *2011 Australian census*) and 20.75% (for *Dbpedia*) of the raw size. In the first case, it even surpasses the effectiveness of `gzip`, being comparable in all datasets. This is a very significant achievement because it demonstrates that these techniques can represent the vocabulary in a space close to that used by a universal compressor and also provide `locate` and `extract` operations.

Note that the `FMI` technique is less effective for URIs. `FMI-RRR` always obtains more compressed representations than `FMI-RG`, being `FMI-RRR` with sampling $s = 128$, and `FMI-RG` with sampling $s = 4$, the best and worst cases respectively. Thus, the range of compression ratios presented in the table corresponds to the `FMI-RRR` variant.

This analysis for URIs can be extrapolated to the **Bnodes** vocabulary, presented in *Dbtune* and the *2000 US Census*. However, a less clear situation arises for **Literals**. As can be seen, `FMI` is the best choice for `Jamendo` and `Dbpedia`, whereas `Re-Pair` is the most effective for the other datasets. Nevertheless, the effectiveness of `FMI` is the most uniform. Experiments show that `FMI-RRR` largely outperforms `FMI-RG`, and larger sampling values improve compression in both cases. In turn, `PFC` and `Hash` obtain very poor results for literals. This fits our initial expectations: in general, literal vocabularies show less regularities than URIs or Bnodes, hence their poor compression ratio. Nonetheless, some particular cases such as the two census datasets and *AEMET* also present regularities in literals (repetition of words or literal tags) which is clearly exploited by `Re-Pair`.

These results entail several remarks. URIs and Bnodes can be highly compressed, being `HTFC` and `Re-Pair` the most effective techniques. However, the compression of literals becomes more complicated as they can contain any type of information. In this case, a prefix-based compression is not always sufficient and a general technique, such as `FMI`, arises as an interesting solution. In fact, `FMI-RRR` outperforms `HTFC` in most cases. As shown, `Hashing` is clearly discouraged when compact representations are required. Finally, note that the classic Front-Coding (`PFC`) achieves limited success, but we will show below that it excels in query performance.

Querying. Next, we evaluate the performance of the `locate` and `extract` operations on the considered techniques. To do so, we design specific micro-benchmarks for testing querying operations on each vocabulary partition: i) `locate` is studied through a batch of 10,000 terms randomly chosen for each vocabulary, and ii) another batch containing 10,000 random IDs are used for `extract`. We run 50 independent executions of each batch and average total *user* times to isolate our measurements of external events. These averaged times per batch are then divided by the number of queries (10,000) to obtain the time per query.

The results for *Dbtune* and the *2000 US Census* are reported in Figures 10.8 and 10.9 respectively. The graphics compare `locate` (left) and `extract` (right) performance for the **URI** (upper), **blank nodes** (middle) and **literal** (bottom) vocabularies. Each graphic draws compression ratios on the X axis

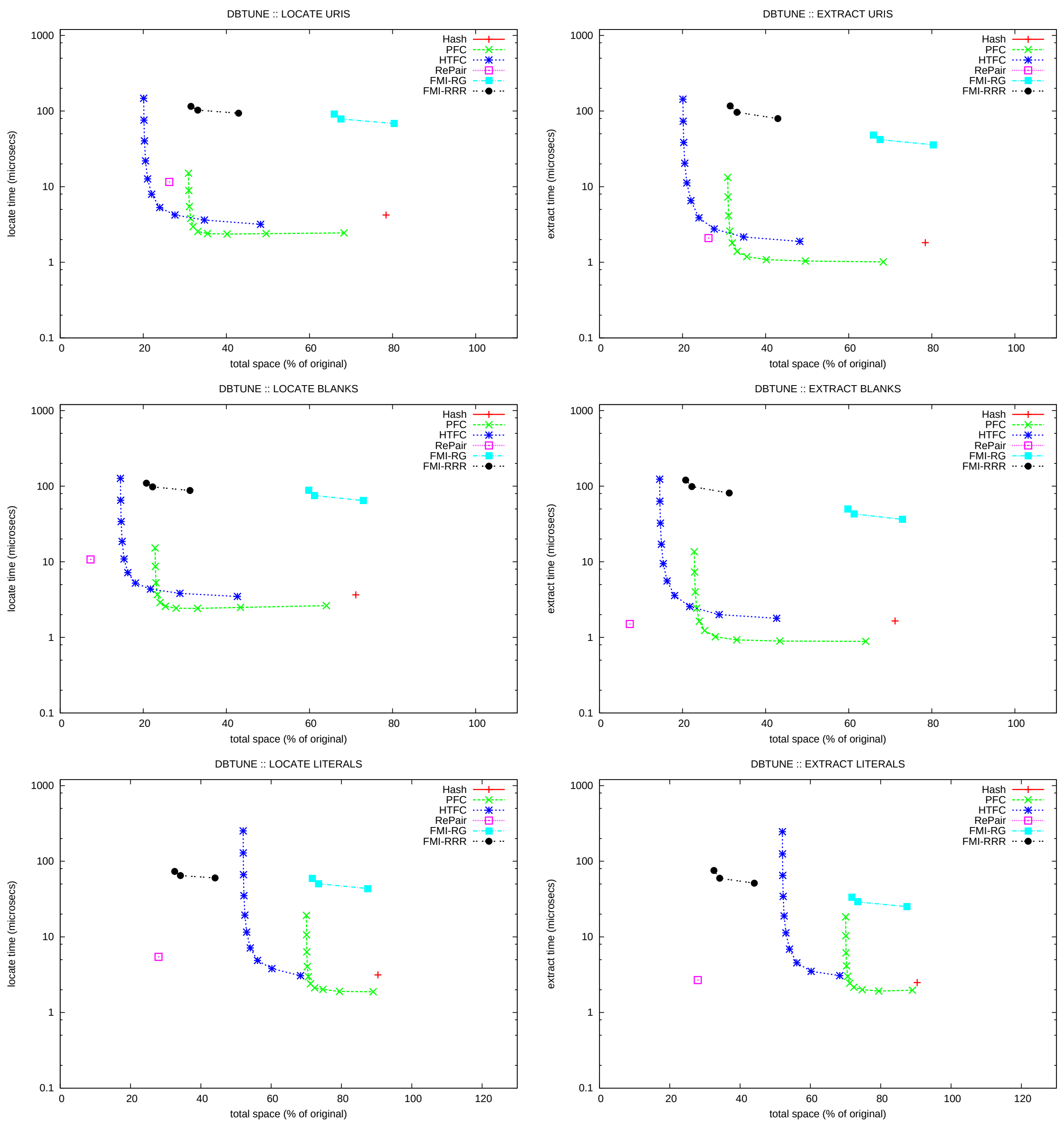


Figure 10.8: `locate` and `extract` times for *URIs* (top), blanks (middle) and *literals* (bottom) of *Dbtune*.

and querying times (in $\mu s/query$) on the Y axis (in logscale). All the conclusions below can be extended to the other datasets in the current setup.

A general conclusion is achieved from all the graphics: the space/time tradeoffs for `Hash` are never the best choice, neither for compression (as shown in Table 10.2) nor at querying times. In general terms, we can state that hashing is not an option for representing RDF dictionaries at large scale. Nonetheless, this compressed hashing technique could always be a choice in simple scenarios without scalability problems: the performance of `locate` and `extract` are around 1-3 $\mu s/query$.

The performance results reported for URIs (top) and blank nodes (middle) are very clear: `PFC` always outperforms `HTFC` in querying because the latter pays the price of the Hu-Tucker decompression. However, as commented above, `PFC` pays a spatial overhead with respect to `HTFC`. With a similar setup,

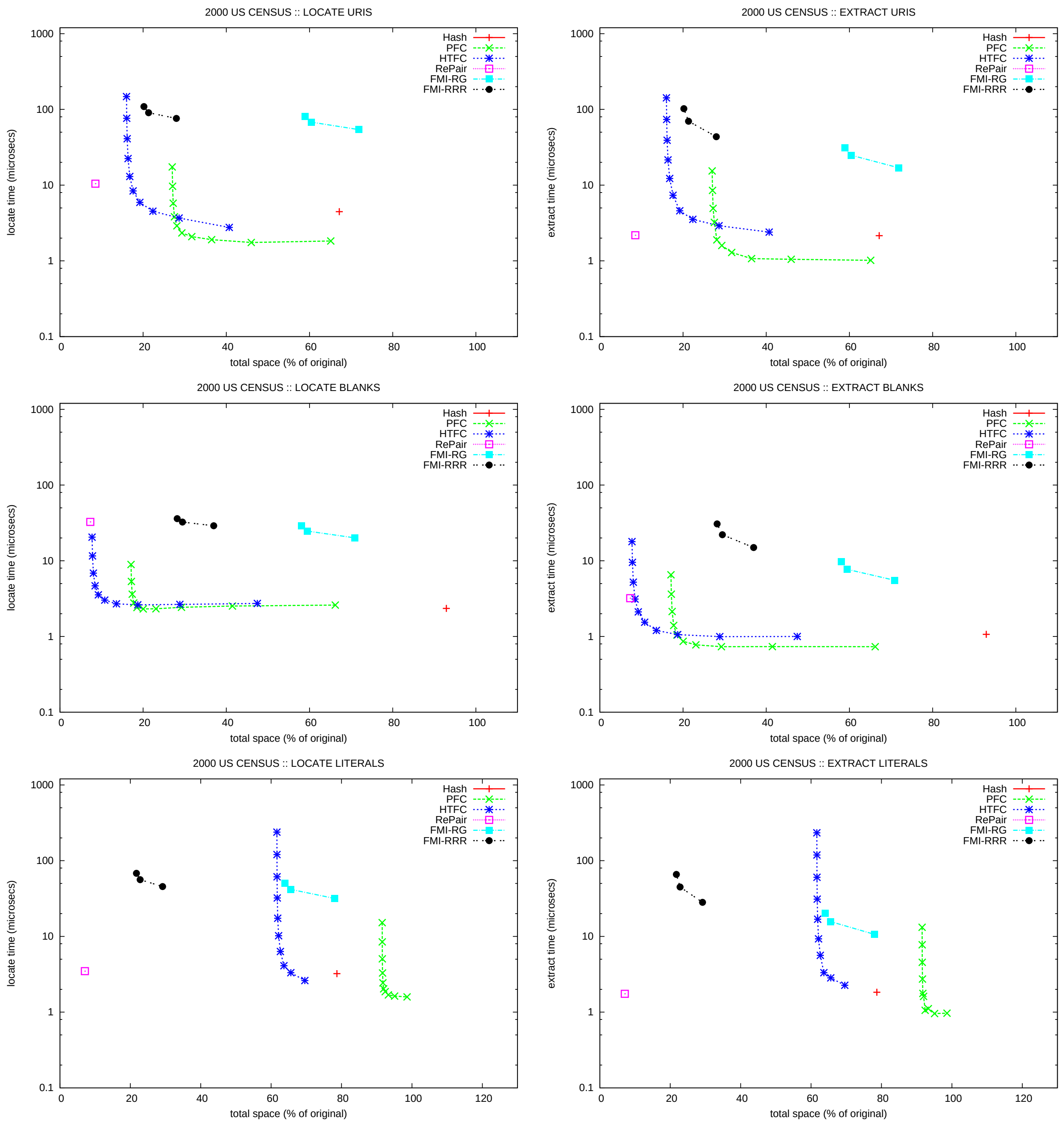


Figure 10.9: `locate` and `extract` times for *URIs* (top), blanks (middle) and *literals* (bottom) of the *2000 US Census*.

the `HTFC` compression ratio is around 8-10 percentage points better than the obtained for `PFC`, but its performance is 1.5 to 10 times slower than `PFC`. The only exception presenting comparable times is the blank nodes performance of the *2000 US Census* (Figure 10.9, middle). Note that the performance difference between `PFC` and `HTFC` is always more noticeable in `extract`. In this operation, all the strings have to be decompressed up to the position of the desired string.

Regarding `Re-Pair`, which presented very competitive compression ratios, its `locate` performance shows a clear degradation with respect to the other techniques, particularly in comparison with `PFC`. In contrast, `Re-Pair` remains competitive in `extract` except for the blank nodes performance of the *2000 US Census*.

Thus, `HTFC` and `Re-Pair` are well-suited for scenarios focused on compression, but `PFC` is the

Dataset	s_r (MB)	**RDF3X**	$\mathcal{D}^{(C)}_{comp}$	$\mathcal{D}^{(Q)}_{comp}$
2011 Australian Census	3.05	147.98%	8.54%	17.52%
Jamendo	31.26	133.66%	18.55%	37.01%
AEMET	81.27	148.33%	14.25%	44.69%
Dbtune	984.66	145.15%	20.73%	43.16%
2000 US Census	551.08	230.54%	13.54%	28.96%
Dbpedia	8098.42	130.71%	30.32%	64.11%

Table 10.3: Compression results of $\mathcal{D}_{comp}$ versus RDF3x dictionaries.

best choice if spatial requirements are slightly relaxed. Finally, note that `FMI` performance is never competitive for URIs.

The analysis for literals is, again, more complex. `PFC` achieves excellent times ($1-10\mu s$/query), but its space is $3-10$ times larger than that used by the most effective techniques, `FMI-RRR` and `Re-Pair`. In turn, `HTFC` largely improves `PFC` compression, but querying times evolve to $3-7$ μs/query for competitive tradeoffs. Finally, `FMI` takes between 25 and 75 μs per query. Thus, `Re-Pair` achieves the most competitive tradeoffs for literals (although the considered implementation suffers from scalability problems in large corpora). In general, `FMI-RRR` and `Re-Pair` must be chosen for optimizing space, but `PFC` may be the option in scenarios where time prevails. Nevertheless, note that `FMI` is still the preferred choice when more sophisticated queries (such as substring-based ones) are desired (Brisaboa et al., 2011). This is consistent with the devised SPARQL filter resolution.

10.5.2 $\mathcal{D}_{comp}$ Performance

As explained above, two functional configurations for $\mathcal{D}_{comp}$ are evaluated. Based on our previous evaluation, we choose those techniques (and their setup parameters) optimizing either the query performance or the dictionary size. In both cases, we keep a competitive space/time tradeoff. The resultant configurations are as follows:

- $\mathcal{D}^{(C)}_{comp}$ is optimized for compression. It implements URI and blank node dictionaries on `HTFC` ($b = 16$), and represents literals with `FMI-RRR`, sampling $s = 128$.

- $\mathcal{D}^{(Q)}_{comp}$ is optimized for querying. It implements URI and blank node dictionaries on `PFC` ($b = 8$), and represents literals with `FMI-RG`, sampling $s = 4$.

Table 10.3 shows compression effectiveness for $\mathcal{D}_{comp}$. In this case, the raw size of the dictionaries (column s_r) considers the raw dump of all $\mathcal{D}_{comp}$ partitions (§10.2). We also include the sizes of the dictionaries used in RDF-3X (Neumann & Weikum, 2010) to compare our results with respect to a real-world solution (note that we measure the space that $\mathcal{D}_{comp}$ takes in memory, but RDF-3X size is measured on disk).

As can be seen, our configuration aimed at compression, $\mathcal{D}^{(C)}_{comp}$, takes more than half of the space used by the configuration optimized for query performance, $\mathcal{D}^{(Q)}_{comp}$. This difference allows for some other configurations whose size can be tuned in accordance to specific application requirements. The comparison of our two variants with respect to RDF-3X gives a magnitude of our achievements with regard to the representation of RDF dictionaries. Whereas RDF-3X always uses more space than the original raw dictionary, our worst $\mathcal{D}^{(Q)}_{comp}$ result for the large `Dbpedia` dataset uses 64.11% of the original space, while the best one for $\mathcal{D}^{(C)}_{comp}$ is only 30.32%. Thus, $\mathcal{D}_{comp}$ reduces the space taken by RDF-3X between 2 and 18 times for the studied datasets.

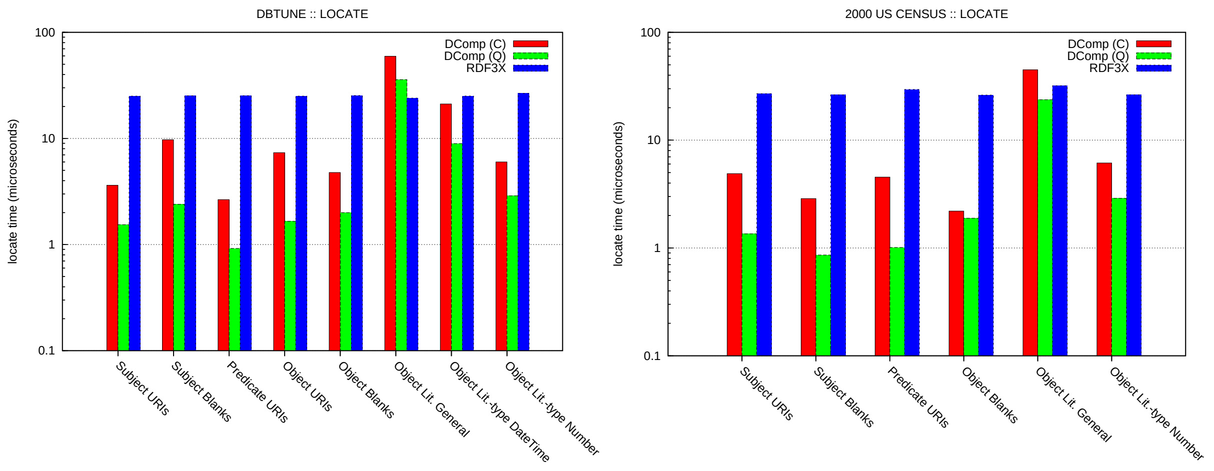


Figure 10.10: `locate` times of $\mathcal{D}_{comp}$ versus RDF3x dictionaries, in *Dbtune* (left) and the *2000 US Census* (right). Y-axis is represented in logarithmic scale.

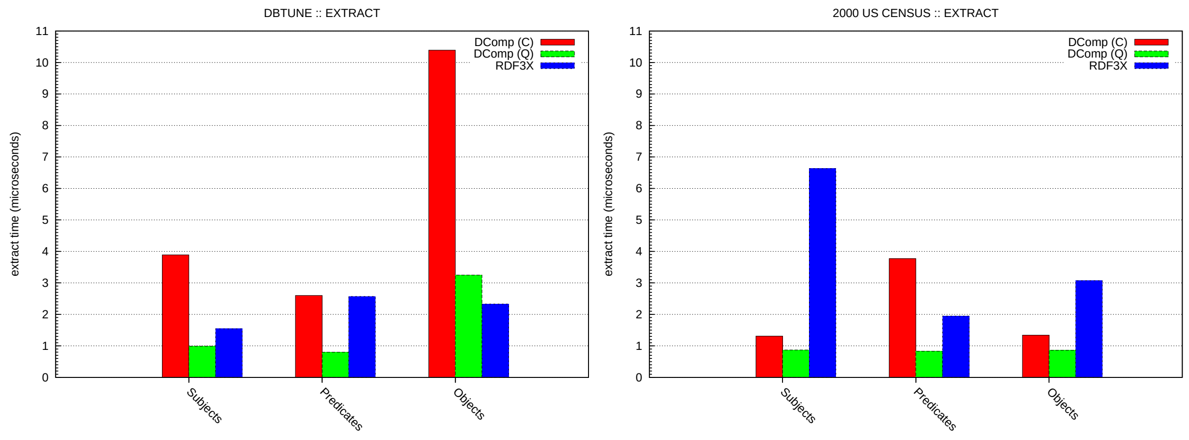


Figure 10.11: `extract` times of $\mathcal{D}_{comp}$ versus RDF3x dictionaries, in *Dbtune* (left) and the *2000 US Census* (right).

These results guarantee that $\mathcal{D}_{comp}$ can be finely tuned to achieve highly-compressed dictionaries. This saves processing resources and enables larger dictionaries to be managed in a fixed main memory. Next, we study $\mathcal{D}_{comp}$ efficiency at querying.

We design a similar random corpora of `locate` and `extract` queries, following the procedure used in the previous experiments. In this case, for `locate` performance evaluation, we consider a batch of $10,000$ random queries for each subdictionary of $\mathcal{D}_{comp}$, and average it in 50 independent repetitions. Figure 10.10 shows the `locate` times in *Dbtune* (left) and the *2000 US Census* (right) compared to RDF3X (note that the y-axis is in logarithmic scale). As can be seen, $\mathcal{D}^{(Q)}_{comp}$ always outperforms $\mathcal{D}^{(C)}_{comp}$, achieving significant differences. It is worth noting that times obtained by our two $\mathcal{D}_{comp}$ configurations are always less than $10\mu s$ per query except for literals. In this case, the use of `FMI`, a more general representation, slightly reduces the performance obtained by the other techniques.

The RDF-3X performance is also analyzed. To do so, we run the query batch and measure the performance time in two different scenarios: "cold" (no data is preloaded in the system main memory) and "warm" (each query is run 5 times prior to the final measure, hence the required data are available in main memory). The comparison is unfair in the cold scenario because RDF-3X needs data to be transferred from disk; these operations are performed in some milliseconds (one order of magnitude

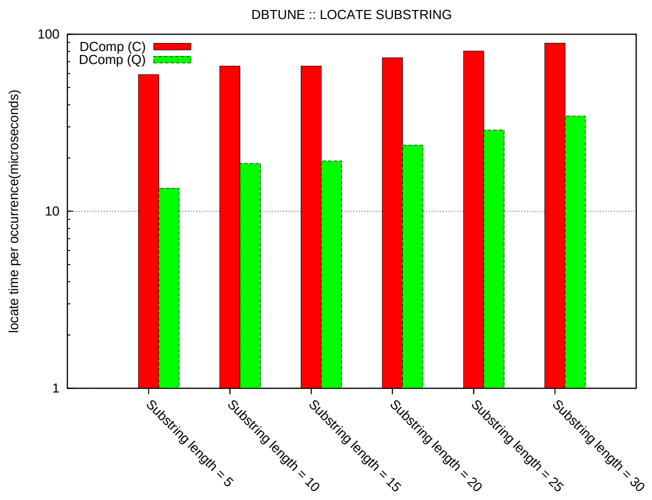


Figure 10.12: $\mathcal{D}_{comp}$ `locate` time per occurrence of the substrings (lengths 5, 10, 15, 20, 25 and 30).

above our technique). Thus, the graphics always report a warm scenario which reduces the times to the level of microseconds. As can be seen, RDF3X performance never improves our approaches for `locate`, except for general literals. Note that, RDF3X is unable to handle tagged literals (it erases the tags when loading the dataset), whereas our approaches give specific support for them.

In turn, we evaluate `extract` performance in Figure 10.11. In this case, we design similar microbenchmarks of 10,000 random queries but restricted to each role: Subjects, predicates and objects. This way, we emulate a real-world extraction of SPARQL results, in which the ID-term solution and its role are known. Results show that `extract` is faster than the previous `locate` operation in all cases. As expected, $\mathcal{D}^{(Q)}_{comp}$ remains faster than $\mathcal{D}^{(C)}_{comp}$, obtaining around $1\mu s$ per query except for the *Dbtune* literals (that are potentially long). This is the only case in which RDF3X outperforms $\mathcal{D}^{(Q)}_{comp}$. Finally, note that $\mathcal{D}^{(C)}_{comp}$ is competitive for `extract`, although the performance in general literals of *Dbtune* is degraded for the same reason.

Thus, in general terms, we can state that $\mathcal{D}^{(Q)}_{comp}$ achieves the best performance for the most used operation in SPARQL engines, remaining highly compressed. In addition, its space/time tradeoff can be finely tuned, bringing it closer to $\mathcal{D}^{(C)}_{comp}$ if compression requirements prevail.

10.5.3 $\mathcal{D}_{comp}$ Regex Resolution

As stated, the `FMI` self-index could not excel for literal compression, but it is the choice for resolving sophisticated queries such as the required for SPARQL *regex* resolution. Thus, we aim at evaluating the performance of substring retrieval on the $\mathcal{D}^{(Q)}_{comp}$ and $\mathcal{D}^{(C)}_{comp}$ configurations. We design a batch of random substring queries of the *Dbtune* literals. To do so, we randomly choose 2,000 substrings of length 5, 10, 15, 20, 25 and 30, and we perform substring `locate` using the `FMI` functionality.

Figure 10.12 shows the results of the evaluation (in microseconds per occurrence of each substring in the dictionary). As can be seen, $\mathcal{D}^{(Q)}_{comp}$ clearly outperforms the $\mathcal{D}^{(C)}_{comp}$ configuration. $\mathcal{D}^{(Q)}_{comp}$ performs significant fast, in the range [13-35] μs per occurrence. As expected, the larger is the substring pattern, the more time is needed to locate. This remains a direct consequence of the `FMI` operation, which performs the pattern matching character by character.

Finally, it is worth noting that this result is obtained with an `FMI` internal sampling of suffixes, by means of a bitmap index. In the current implementation, this bitmap is compressed with a `RRR` configuration and parameter 64. Figure 10.13 shows the performance considering other configurations

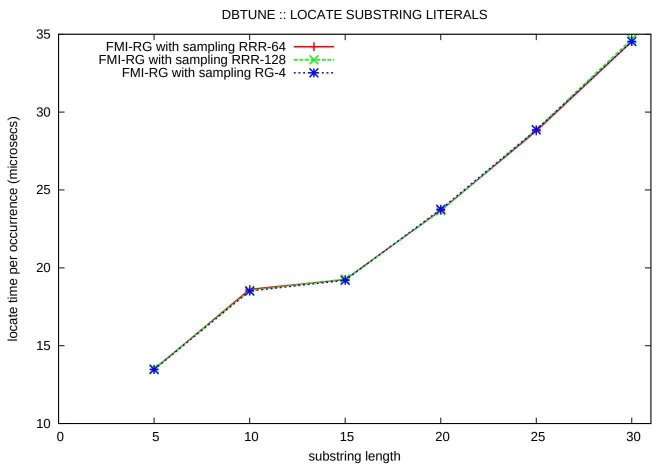


Figure 10.13: $\mathcal{D}_{comp}$ `locate` time of substrings (lengths 5, 10, 15, 20, 25 and 30) with different `FMI-RG` samplings, showing a similar performance.

for this sampling. In particular, we evaluate the same $\mathcal{D}_{comp}^{(Q)}$ configuration with a sampling bitmap `RG` (parameter=4), `RRR` (parameter=64) and `RRR` (parameter=128). The graphic reports that, although the sampling with a bitmap `RG` is slightly faster, the difference is not significant. In addition, the size of the literal dictionary with the `RG` sampling becomes 9% and 10% more than the `RRR` alternatives. Nevertheless, note that this time is given “per occurrence” and, thus, this sampling could be an important tradeoff to take into account if resolution time prevails. Finally, note that, in all the alternatives of Figure 10.13, the `locate` times present a linear growth *w.r.t* the length of the substring patterns.

It's not a question of where he grips it! It's a simple question of weight ratios! A five ounce bird could not carry a one pound coconut.

Monty Python and the Holy Grail (1975)

11 Discussion

This chapter briefly summarizes the contributions (§11.1) of this part of the thesis devoted to RDF dictionaries. We also devise future work and applications (§11.2).

11.1 Contributions

Through the previous chapters, we have addressed compressed representations for RDF dictionaries. First of all, in Chapter 9, we have introduced the problem of effective representations of RDF dictionaries in the novel scenario of Big Semantic Data. We stated that current techniques used for their representation suffer from scalability issues. We then reviewed existing techniques for compressed string dictionaries.

Next, in Chapter 10, we applied these techniques to the specific case of RDF and obtained simple compressed representations for URI, blank node and literal dictionaries. This experience was integrated within a novel RDF dictionary, called $\mathcal{D}_{comp}$, which addresses specific management of compressed RDF dictionaries. $\mathcal{D}_{comp}$ reorganizes the RDF dictionary into subdictionaries according to its role and term vocabulary, allowing for specific compression of each part. We detailed its data structures and algorithms able to perform typical querying (`locate` and `extract`) and we introduced advanced filter resolution leveraging $\mathcal{D}_{comp}$ features.

Finally, we performed a deep evaluation on real-world datasets, showing many interesting remarks:

- The application of the techniques from compressed string dictionaries to RDF dictionaries is able to achieve high compression ratios. These ratios are, in general, comparable to that achieved by universal compressors. In addition, these techniques provide query functionality (`locate` and `extract` operations) on compressed space.

- Traditional hashing is discouraged for large RDF vocabularies as its space/time tradeoff is never the best choice. Nonetheless, it presents comparable performance, remaining a simple solution for basic scenarios.

- A prefix-based compression, `HTFC`, and a grammar-based compression, `Re-Pair`, are the most effective techniques for compressing URIs and blank nodes. `Re-Pair` and the `FM-Index` self-index, are the best techniques for compressing literals.

- $\mathcal{D}_{comp}$ achieves highly-compressed dictionaries, between $9-64\%$ of its original size, and it excels in query performance, answering queries in $1-60\mu s$.

- Compared to the $B^{+}-tree$ proposal in RDF-3X, $\mathcal{D}_{comp}$ reduces its space between 2 and 18 times. $\mathcal{D}_{comp}$ performs significant faster except for literals, being comparable in this case.

- The `FM-Index` self-index used for literals in $\mathcal{D}_{comp}$ resolves SPARQL *regex* queries in the range of microseconds per occurrence (with a linear growth *w.r.t* the length of the substring pattern).

All this experience guarantees that $\mathcal{D}_{comp}$ is an innovative technique which i) achieves highly-compressed dictionaries, ii) is highly parameterizable, allowing to configure its space/time tradeoff, iii) achieves an extraordinary performance for the most used dictionary operations in SPARQL engines; and iv) opens up further optimizations for filter resolution.

11.2 Future Work and Applications

As stated, our proposal $\mathcal{D}_{comp}$ perfectly fits the philosophy of `HDT`, and it can be directly plugged as the representation of the dictionary component. However, we have shown that RDF dictionaries are commonly used in all kind of applications performing on the Web of Data. Our future work focuses on elaborating a toolkit of RDF dictionaries and a set of best practices to different domains. The idea is to be able to easily integrate $\mathcal{D}_{comp}$ as a dictionary index within an existing application performing on Big Semantic Data, such as a SPARQL engine. A (semi-) automatic analysis of the type of data and the expected functionality would help recommend the correct parametrization.

Besides, the use of $\mathcal{D}_{comp}$ provides interesting features for filtering which can be further exploited. First, a line of future work is to integrate $\mathcal{D}_{comp}$ features into a SPARQL query planner to fully exploit its organization and characteristics, in particular for early filter evaluation. In turn, the *regex* resolution in compressed space opens up many interesting applications. In this respect, we have managed multimedia metadata (Arias, Corcho, Fernández, Martínez-Prieto, & Suárez-Figueroa, 2013) in practice: we made use of $\mathcal{D}_{comp}$ (and `HDT`) to provide full-text search of multimedia metadata in compressed space. Performance experiments reported that our solution overcame Virtuoso for all queries in the setup (see Arias et al. (2013) for further details).

Finally, an additional line of future work focuses on evolving $\mathcal{D}_{comp}$ to support dynamic operations of insert, delete, and update. These operations are essential to integrate $\mathcal{D}_{comp}$ in semantic databases in which dictionaries evolve according to triples management. Nevertheless, we devise two lines of work. On the one hand, one could work on making the original techniques from compressed string dictionaries also dynamic. This would help in the $\mathcal{D}_{comp}$ evolution, although some problems (such as the efficient movement of terms between subdictionaries), should be treated aside. On the other hand, $\mathcal{D}_{comp}$ modification could be studied as a subproblem of the alteration of Big Semantic Data. In other words, one could work in infrastructures allowing to modify Big Semantic Data, and incorporate the dictionary component as a problem to contend with. We will provide some notes on these potential infrastructures in future chapters.

Part IV

Compact RDF Triple Indexes

You're gonna need a bigger boat.
Jaws (1975)

12 Introduction

This part of the thesis completely focuses on triple indexes. The current chapter first motivates the scalability problems of RDF triple indexes (§12.1), and reviews the state-of-the-art techniques (§12.2). Their drawbacks help define our goals (§12.3) which mainly comprises the design of compact triple indexes on top of HDT-encoded datasets.

12.1 Motivation

The ID-triples in an HDT-encoded dataset can be directly accessed once its components are loaded into the memory hierarchy of any computational system. As we review below (§12.2), the state of the art in ID-triple indexes is large. Nonetheless, we first highlight some interesting remarks pointed out throughout this thesis.

First of all, some consumption processes does not need to resolve complex SPARQL queries, but a minimum set of operations. For instance, if the entity $< e >$ is dereferenced in accordance to the third Linked Data principle, one should perform a query such as:

CONSTRUCT $\{ < e >$ *?predicate ?object* . $\}$
WHERE$\{ < e >$ *?predicate ?object* . $\}$

or, at most, one could also retrieve all the resources pointed to this same entity (if $< e >$ is playing as an object), or the triples in with $< e >$ is playing as an object (if exist). Then:

CONSTRUCT $\{ e$ *?predicate ?object* .
?subjectP $< e >$ *?objectP* . $\}$
?subjectO predicateO $< e >$. $\}$
WHERE$\{ < e >$ *?predicate ?object* . $\}$
UNION$\{$ *?subjectP* $< e >$ *?objectP* . $\}$
UNION$\{$ *?subjectO predicateO* $< e >$. $\}$

We can find another simplified scenario in the area of RDF streaming, in which a limited set of queries is repeated over a stream of data. Thus, in some scenarios such as the presented above it is interesting to provide small indexes with partial functionality regarding SPARQL. This was also pointed out in `HDT triples encoding`, classifying four levels of triple functionality (§7.1.3). However it is clear than state-of-the-art indexes are guided by an intensive-querying perspective, corresponding to *L2-Join Resolution* or *L3-Full Sparql* levels.

In addition, RDF indexes suffer from scalability issues in Big Semantic Data as they barely address compression notions. The space optimization achieved in HDT-encoded datasets is not fully exploited if it is then loaded into burdensome structures. In contrast, one could argue that succinct data structures could be added to the HDT-encoded dataset in order to provide direct access to the triple information. Moreover, if complex SPARQL resolution is needed, advanced succinct indexes could also be constructed on top of the HDT-encoded dataset. In other words, the HDT-encoded data is not parsed to pre-built RDF indexes, but RDF indexes are built for the HDT-encoded data at consumption time.

This goal is detailed at the end of this chapter (§12.3), and developed in the next chapter. We first review the state of the art in RDF triple indexes and stores.

12.2 State of the Art

Several RDF indexes and RDF stores explore efficient RDF retrieval and SPARQL resolution. As RDF does not prevent any technique, the implementation of these proposals has a direct effect on the retrieval efficiency, and therefore on the success of SPARQL-based solutions in the Web of Data. We review below the existing techniques for modeling, partitioning, and indexing RDF, and discuss their use in some real RDF stores.

Although the explosion of novel RDF engines could make this review uncompleted, we focus on showing the main achievements and shortcomings in the state of the art. We first summarize those approaches (and their techniques) based on a relational infrastructure. Then, we show solutions natively designed for RDF.

12.2.1 Relational Solutions

Some logical schemes have been proposed for representing RDF over the infrastructure provided by relational databases. Although they leverage the “strictness” of the relational model for handling the semi-structured RDF features, there is still room for optimizations (Sakr & Al-Naymat, 2010). We describe below the most used schemes.

Single three-column table. This is the most straightforward scheme modeling RDF over a relational infrastructure. It represents RDF as a huge single table of three columns, holding an RDF triple (S,P,O). Systems such as 3store (Harris & Gibbins, 2003) or the popular Virtuoso[1] implement this scheme.

Virtuoso (Erling & Mikhailov, 2007, 2009) is probably the main representative of these approaches. In particular, Virtuoso extends each triple in the three-column table with an additional column holding the graph (G) it belongs[2]. To minimize redundancy, Virtuoso makes use of an RDF dictionary, hence the S, P, O and G columns store IDs. Another (dictionary) table holds the mapping between each ID and the term representing it from the subject, predicate, object or graph vocabulary.

As can be seen, SPARQL resolution involves many expensive self-joins on the huge single four-column table. Nevertheless, Virtuoso uses two indexes: (G,S,P,O) and (O,G,P,S). From version 6, it also includes 3 partial indexes (SP, OP and GS). Note that, if there are frequent updates, keeping this amount of indexes fresh would affect performance. Instead, Virtuoso keeps the full-indexes updated and can i) completely drop and recreate the partial indexes or even ii) disable the use of partial indexes for intensive updates. In any case, Virtuoso indexes are optimized for workloads of bulk-load and read-intensive access patterns with few deletes[3].

Several compression techniques are considered. First, each database page store only distinct values and, then, gzip is applied to these pages (Erling & Mikhailov, 2009). In addition, from version 7, indexes are column-wise stored (Erling, 2012), saving 1/3 of space. This latter version introduces other important improvements to boost query parallelization, such as vectorized execution of queries (Sompolski, Zukowski, & Boncz, 2011). As can be seen in the BSBM evaluation, Virtuoso (version 7) is one of the most scalable solutions. It excels in performance and remains very competitive in space.

[1]`http://www.openlinksw.com/`

[2]Note that multiple graphs (datasets) can be managed in a single scheme. This fits the notion of N-quads (Carothers, 2013).

[3]See `http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VirtRDFPerformanceTuning`.

Property tables. This model arises as a natural practical scheme for RDF organization in relational databases as it proposes to create relational-like property tables of RDF data. Each table holds multiple predicates (properties) over a list of subjects. Thus, a given property table has many columns as different predicates (one per column) are used for describing the subjects that it stores (in rows). Although this model reduces significantly the number of self-joins, the cost of the query resolution remains high. Besides, the use of property tables induces two additional problems. On the one hand, note that subjects can appear in one table even if some columns (predicates) are missed. In other words, storage requirements increase because NULL values must be explicitly stored if the represented subject is not described for a given property in the table. On the other hand, multi-valued attributes are abundant in semantic datasets and they are somewhat awkward to express in property tables (Abadi et al., 2007). Thus, property tables are a competitive choice for representing well-structured datasets, but they lose potential in a general case. Systems like Jena (Wilkinson, 2006; Wilkinson et al., 2003) or Sesame (Broekstra, Kampman, & van Harmelen, 2003) use property tables for modeling RDF.

Jena TDB[4] is a persistent module to implement a high performance RDF store for the in-memory Jena. A dataset is stored in persistent data structures by a custom implementation of threaded B+Trees for triples and triples plus graphs. A dictionary of ID-terms is also used. In this case, the ID is a hash of the term (a 128 bit MD5 hash), indexed by a B+Tree.

Of all Sesame-based implementations, we highlight BigOWLIM (recently renamed OWLIM-SE) belonging to the family of OWLIM native semantic repositories (Bishop et al., 2011). It is a commercial Java implementation designed as a database management system implementing the Sesame's SAIL APIs. BigOWLIM holds two main indexes, (P,O,S) and (P,S,O). It can also enable indexes by graph (context) and partial indexes similar to Virtuoso. In addition, the data on disk can be compressed using ZIP with a compression parameter to manage the space/time tradeoff.

Vertical partitioning. The vertical partitioning (VP) scheme (Abadi et al., 2007) is based on the fact that few predicates are used to describe a dataset. This way, VP uses many tables as different predicates are used in the dataset, each one storing tuples `(S,O)` that represent all (subject,object) pairs related through a given predicate. Each table is sorted by the subject column, in general, so particular subjects can be located quickly, and fast merge joins can be used to reconstruct information about multiple properties for subsets of subjects (Abadi et al., 2007). In the absence of other indexes, though, this decision penalizes queries by object.

Nevertheless, the main weakness of VP-based solutions is the lack of efficiency for queries with unbounded predicates. In this case, all tables must be queried and their results must then be then merged to obtain the final results. This cost increases linearly with the number of different predicates used in the dataset. Thus VP is not the best choice for representing datasets with many predicates, unless other partial indexes are used.

In contrast, VP-based solutions avoid the weaknesses previously reported for property tables because only non-NULL values are stored, and multi-valued attributes are listed as successive tuples in the corresponding table. Moreover, VP can be perfectly used in combination with column-oriented databases.

Abadi et al. (2007) and Abadi, Marcus, Madden, and Hollenbach (2009) report that querying performance in column-oriented databases is up to one order of magnitude better than that obtained in row-oriented ones. This fact motivates the implementation of their system SW-Store as an extension of the column-oriented database C-Store (Stonebraker et al., 2005). SW-Store leverages all the advantages reported above, but also suffers from a lack of scalability for queries with unbounded predicate. SW-Store, also perform a dictionary encoding that maps long URIs and literal values to integer IDs. In addition to the VP scheme, SW-Store also indexes some materialized path expressions. This speeds up path expressions resolution at the price of increasing storage requirements. Sidirourgos et al. (2008) show additional

[4] `http://jena.apache.org/documentation/tdb/index.html`

experiments on VP. They replace C-Store by MonetDB[5] in the database layer; these systems show a couple of differences (Schmidt, Hornung, Küchlin, Lausen, & Pinkel, 2008): i) data processing in C-Store is disk-based while it is memory-based in MonetDB; and ii) C-Store implements carefully optimized merge joins and makes heavy use of them, whereas MonetDB uses merge joins less frequently. Even so, MonetDB arises as a competitive choice in this scenario (Sidirourgos et al., 2008). The findings reported in these works differ from each. Whereas Abadi et al. (2007) and Abadi et al. (2009) conclude that VP overcomes property tables, Sidirourgos et al. (2008) refute this conclusion and show that the comparison depends on the dataset features.

12.2.2 Native Solutions

Native solutions are designed from scratch to better address RDF peculiarities. Although some works (Anglés & Gutiérrez, 2005; Bönström, Hinze, & Schweppe, 2003; J. Hayes & Gutiérrez, 2004) propose different graph-based models, the main line of research focuses on **multi-indexing** solutions. YARS (Harth & Decker, 2005; Harth, Umbrich, Hogan, & Decker, 2007) proposes a six B+-tree indexes for managing N-quads: (S,P,O,C), (P,O,C), (O,C,S), (C,S,P), (C,P) and (O,S). This scheme allows all quads conforming to a given query pattern (in which the context can also be a variable) to be quickly retrieved. This experience has been integrated in many systems within the current state of the art for RDF management. Note also that YARS performs on a dictionary encoding, then the quads (enhanced with the contexts) are regarded as ID groups.

Hexastore (Weiss, Karras, & Bernstein, 2008) adopts the rationale of VP and multi-indexing. In contrast to VP, Hexastore treats subjects, predicates, and objects equally. That is, whereas VP prioritizes predicates and indexes pairs (subject,object) around them, Hexastore builds specific indexes around each dimension and defines a priority between the other two. This way, Hexastore manages six indexes: `(S,P,O)`, `(S,O,P)`, `(P,S,O)`, `(P,O,S)`, `(O,S,P)`, and `(O,P,S)`. In a naive comparison, the VP scheme (sorted by subject) can be seen as an equivalent representation to the index `(S,P,O)` in Hexastore. Thus, Hexastore stores triples in a combination of sorted sequences that requires, in the worst case, 5 times the space used to index the full dataset in a single triples table. This is because some sequences can be shared between different indexes (for instance, the object sequence is interchangeably used in the indexes `SPO` and `PSO`). The Hexastore organization ensures primitive resolution for all triple patterns and also that the first step in pairwise joins can be always implemented as fast merge joins. However, its large storage requirements slow down Hexastore when representing large datasets, because it is implemented as an *in-memory* solution.

RDF3X (Neumann & Weikum, 2010) goes one step further and introduces index compression to reduce the spatial requirements reported above. In contrast to Hexastore, RDF3X creates its indexes over a single "giant triples table" (with columns $\mathtt{v}_1, \mathtt{v}_2, \mathtt{v}_3$), and stores them in a (compressed) clustered B^+-tree. Triples, within each index, are lexicographically sorted[6] allowing SPARQL patterns to be converted into range scans.

The collation order implemented in the RDF3X table causes neighboring triples to be very similar. In most cases, neighboring triples share the values in $\mathtt{v}_1$ and $\mathtt{v}_2$, and the increases in $\mathtt{v}_3$ are very small. This fact facilitates differential compression to represent a given triple with respect to the previous one. This scheme is leaf-oriented within the B^+-tree, so the compression is individually applied on each leaf. Although the authors test some well-known bitwise codes (γ-codes, δ-codes, and Golomb codes (Salomon, 2007b)), they finally apply a bytewise code specifically designed for differential triples compression. This technique ensures highly-efficient decompression with a slight spatial overhead with respect to the most effective codes. Finally, it is worth noting that RDF3X also manages aggregated indexes `(SP)`, `(PS)`, `(SO)`, `(OS)`, `(PO)`, and `(OP)`, which store the number of occurrences of each pair

[5]`http://www.monetdb.org/`

[6]RDF3X also performs dictionary encoding, so the ordering is carried out on the element IDs.

in the dataset. RDF3X also contributes with a RISC-style query processor that mainly relies on merge joins over the sorted indexes. Besides, it implements a query optimizer mostly focused on join ordering in its generation of execution plans.

RDF3X reports a very efficient performance that outperforms SW-Store by a large margin. These results make it a leading reference in the area. However, despite its compression achievements, the spatial requirements in RDF3X remain very high. This involves an indirect overhead to the querying performance because large amounts of data need to be transferred from disk to memory, and this can be a very expensive process with respect to the query resolution itself (Schmidt, Hornung, Küchlin, et al., 2008; Sidirourgos et al., 2008).

BitMat (Atre et al., 2010) follows the idea of managing compressed indexes, but it goes another step further and proposes querying algorithms that directly perform on the compressed representation. BitMat introduces an innovative compressed bit-matrix to represent the RDF structure. It is conceptually designed as a bit-cube $\mathtt{S}\times\mathtt{P}\times\mathtt{O}$, but its final implementation slices to get two-dimensional matrices: `SO` and `OS` for each predicate `P`, `PO` for each subject `S`, and `PS` for each object `O`. These matrices are run-length (Salomon, 2007b) compressed by taking advantage of their sparseness. Two additional bitarrays are used to mark non-empty rows and columns in the bitmats `SO` and `OS`. The results reported for BitMat show that it only overcomes the state of the art for low selectivity queries. However, it is an interesting achievement because it demonstrates that avoiding materializaton of intermediate results is a very significative optimization for these queries.

A novel solution (Tran et al., 2013) explore the RDF structuredness and index groups of predicates and their instantiated data. Its performance for large datasets is still pending.

Finally, we highlight the novel hybrid (Sakr, Elnikety, & He, 2012) and the full in-memory RDF stores (Binna, Gassler, Zangerle, Pacher, & Specht, 2011; Janik & Kochut, 2005) which represent an emerging alternative in this scenario. Nevertheless, their current results are often limited to manage small datasets. Their scalability is clearly compromised by the use of structures, like indexes and hash tables, that demand large amounts of memory. However, some semantic applications, such as *inference-based* ones, claim for scalable in-memory stores because they perform orders of magnitude faster if the entire dataset is in memory (Huang, Abadi, & Ren, 2011), and they also support a higher degree of reasoning.

A recent VP-based approach, called k^2-triples (Álvarez-García et al., 2011), uses compact data structures to compress and index the triples full in-memory. It represents the graph as $|P|$ adjacency matrices of $\mathtt{S}\times\mathtt{O}$ cells. Each matrix is represented with a k^2-tree (Brisaboa, Ladra, & Navarro, 2014), a compact structure leveraging the very sparse 1 distributions to achieve an ultra-compressed representation. A recent improvement, k^2-triples+ (Álvarez-García, Brisaboa, Fernández, Martínez-Prieto, & Navarro, 2013), enhances the vertical partitioning with additional SP and OP indexes. This mitigates the main VP drawback (inefficiency in queries with unbounded predicates) at the cost of a limited space overhead.

New opportunities arise also thank to the advances in distributed computing. This class of solutions, recently studied (Huang et al., 2011; Urbani et al., 2010) on the MapReduce framework, allows arbitrarily large RDF data to be handled because more nodes can be added to a cluster when more resources were necessary. BigData[7] is an horizontally scaled storage inspired by the Google bigtable architecture. It can be deployed in a single machine as well as over a cluster of machines, with a dynamic key-range partitioning. This latter allows to manage larger datasets once the federation can scale incrementally adding new machines without reloading the data.

The underlying RDF representation reflects the YARS2 (Harth et al., 2007) scheme, using three indexes, (S,P,O), (P,O,S) and (O,S,P), scaling up to six if graph (context) information is managed.

BigData uses a concurrency control for readers and writers. Thus, writers are absorbed onto specific pre-sized nodes, which are migrated to optimized read-only B+Tree files when they are filled. Caching is also used to reduce inter-node communications[8].

[7]`http://www.systap.com/bigdata.htm`

[8]More details on specific distributed indexes can be found in `http://www.systap.com/pubs/graph_databases.pdf`.

Nevertheless, these distributed systems still require further research to ensure efficient RDF exchanging (Fernández et al., 2011; Fernández et al., 2013), as well as efficient performance in each node.

In summary, the vast majority of these approaches suffers from lack of scalability (specially noticeable using vertical partitioning (Sidirourgos et al., 2008)), and uses naive compression approaches. There is still a large interest in querying optimization (Schmidt et al., 2010), whose performance is diminished when the RDF stores manage very large datasets.

12.3 Our Goal

We have presented current scalability problems arising in RDF triple indexes for Big Semantic Data. In turn, we have shown that several scenarios require an index structure that keeps the compactness of the encoding providing basic or complex queries over triples. These are, in summary, our main objectives:

- To design **triple indexes on top of HDT-encoded datasets**, leveraging its compression ability.
- To provide a **fast index construction** process at consumption time.
- To allow for **specific tuning** to perform basic or complex queries.
- To provide different **space/time tradeoffs** for different purposes.
- To make use of **succinct data structures** for such indexes, achieving several advantages:
    1. To reduce the size of indexes, thus mitigating scalability problems.
    2. To perform in main memory on large compressed datasets, thanks to the compression and functionality of these succinct data structures.
    3. To take advantage of the memory hierarchy to improve performance time.

Although these indexes are aimed at HDT-encoded datasets, one could argue that a standalone configuration is equally efficient. The only difference lies on the workflow of events. In the original proposal we leverage the previous *exchanged* HDT to build an index on top, “as fast, compressed, and performance efficient as possible” for the scenario one wish to play. In contrast, a standalone configuration stands for an RDF index whose “file system” is HDT, no matter if data is or not exchanged after o before the consumption process.

I am big! It's the pictures that got small.
Sunset Boulevard (1950)

13 Compact RDF Indexes on top of `HDT` Encodings

As stated, several Triples encoding are feasible with different trade-offs between the compression ratio (exchanging) and some natively supported operations over the triples (consumption). We first revisit our `HDT` Triples encoding, proposing a more practical Bitmap Triples (BT) configuration (§13.1). This encodes the structure of the graph in two correlated bitsequences which can be indexed by means of succinct data structures at consumption time (§13.1.2). This provides a basic retrieval feature which can perfectly fit simple scenarios (such as the proposed in the motivation of Section 12.1). Next, we consider the use of additional compressed succinct data structures to resolve all kind of SPARQL triple patterns (§13.2). This sets the basis of full SPARQL resolution. Finally, we experiment the compressibility and query performance of all indexes on a testbed (§13.3).

13.1 `HDT` Bitmap Triples Encoding

As stated, the triples in RDF could be represented as adjacency lists, one per subject. In `HDT`, the proposed Compact Triples encoding used this conceptualization. However, note that an adjacency list can also be seen as a tree, and then an RDF dataset comprises a forest of trees, one per subject. This is represented in the top of Figure 13.1. A graph, then, contains one tree per subject ID, the first S level of the tree. As stated, subject IDs are sequential, hence the first level of subjects is implicitly represented. Then, the second level lists all predicates (also sorted IDs) related to the subject, and finally the leaves organize all objects (sorted) for each pair *(subject, predicate)*.

Compact Triples encoded this conceptualization roughly. The `0`'s were auxiliary values denoting a change of list, hence they represent, implicitly, the tree-shaped structure. However, these values were embedded in each stream, mixing data and structure.

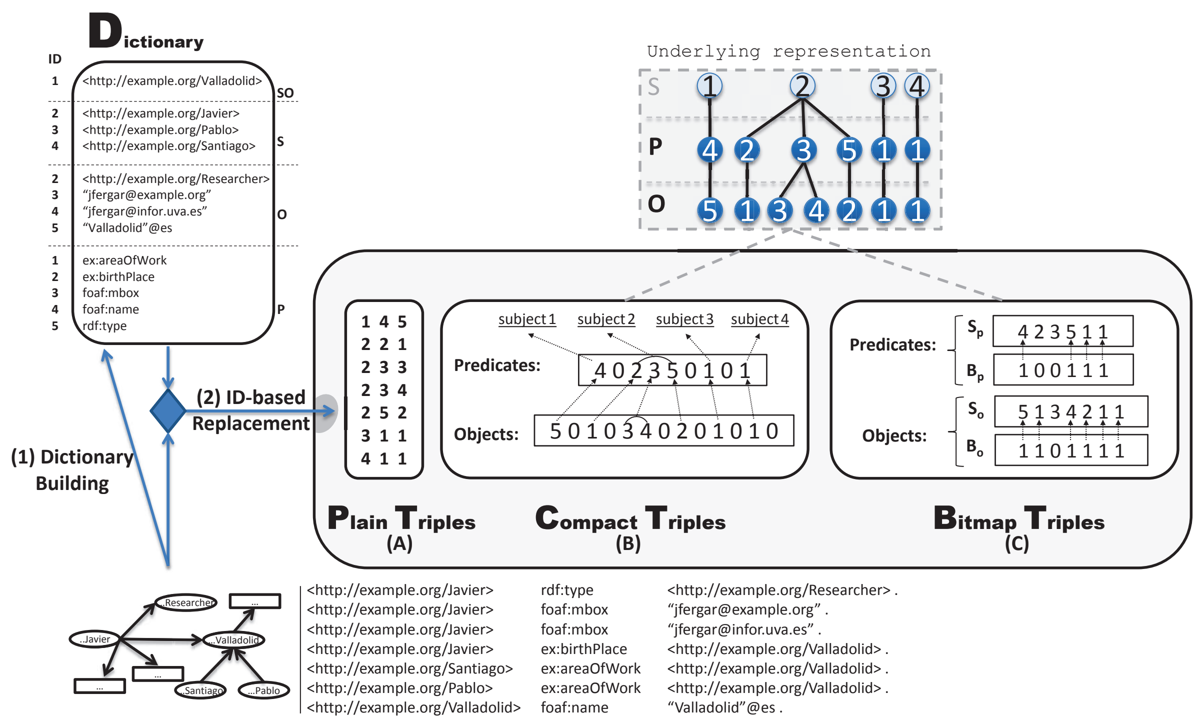
Bitmap Triples (BT) follows this idea and encodes the forest of trees in a more intelligent way, shown in 13.1(C). First, subjects are again implicitly represented. Two structures are then used for predicates:

- An ID sequence ($\mathcal{S}_p$) concatenates predicate lists following the tree ordering.
- A bitsequence ($\mathcal{B}_p$) uses one bit per element in $\mathcal{S}_p$: `1`-bits mean that this predicate is the last one for a given tree, whereas `0`-bits are used for the remaining predicates.

For instance, in Figure 13.1(C), the second `1`-bit in $\mathcal{B}_p$ marks the end of the predicate adjacency list for the second subject which is $\{2, 3, 5\}$.

In turn, object encoding is performed in a similar way:

- An ID sequence ($\mathcal{S}_o$) concatenates object lists following the tree ordering.
- A bitsequence ($\mathcal{B}_o$) uses one bit per element in $\mathcal{S}_o$: `1`-bits represent the last object related to the corresponding *(subject,predicate)* pair, and `0`-bits the remaining ones.



<http://example.org/Javier>	rdf:type	<http://example.org/Researcher> .
<http://example.org/Javier>	foaf:mbox	"jfergar@example.org" .
<http://example.org/Javier>	foaf:mbox	"jfergar@infor.uva.es" .
<http://example.org/Javier>	ex:birthPlace	<http://example.org/Valladolid> .
<http://example.org/Santiago>	ex:areaOfWork	<http://example.org/Valladolid> .
<http://example.org/Pablo>	ex:areaOfWork	<http://example.org/Valladolid> .
<http://example.org/Valladolid>	foaf:name	"Valladolid"@es .

Figure 13.1: The proposed practical HDT triple encodings.

In Figure 13.1(C), 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 related to the *(subject,predicate)* pair $(2, 3)$.

Note also that the characterization of the lists with the proposed metrics remains valid for Bitmap Triples as well as Compact Triples (see characterization in §7.2.3). In short:

- The length of predicates for a given subject i in $\mathcal{S}_p$ is exactly its labeled out-degree, $degL^-(i)$.
- The expected mean and maximum length of the predicates lists in $\mathcal{S}_p$ are given by $\overline{degL^-}(G)$ and $degL^-(G)$ respectively. Remember that the empirical evaluation in Section 4.3.3 shows that few predicates are related to the same subject, less than 20 on average.
- For a given $(subject, predicate)$ pair, (i, j), its partial out-degree, $deg^{--}(i, j)$, denotes the size of the corresponding list in $\mathcal{S}_o$.
- The expected mean and maxim values of the object lists in the $\mathcal{S}_o$ are given by $\overline{deg^{--}}(G)$ and $deg^{--}(G)$ respectively. The evaluation in Section 4.3.4 states that the mean partial out-degree is slightly bigger than 1, which implies short object lists for each (subject,predicate) pair.

13.1.1 BT Conceptual Navigability

The proposed HDT Bitmap Triples encoding (§7.2.3) allows the RDF graph to be largely compressed by isolating ID-terms from the link structure, which is represented in two coordinated bitmaps. Thus, these bitmaps are the core for accessing and querying the RDF graph at consumption.

For illustration purposes, we show in Figure 13.2 an excerpt from the previous example in Figure 13.1. As can be seen, adjacency lists draw tree-shaped structures containing the subject ID in the root, the predicate IDs in the middle level, and the object IDs in the leaves (note that each tree has as many

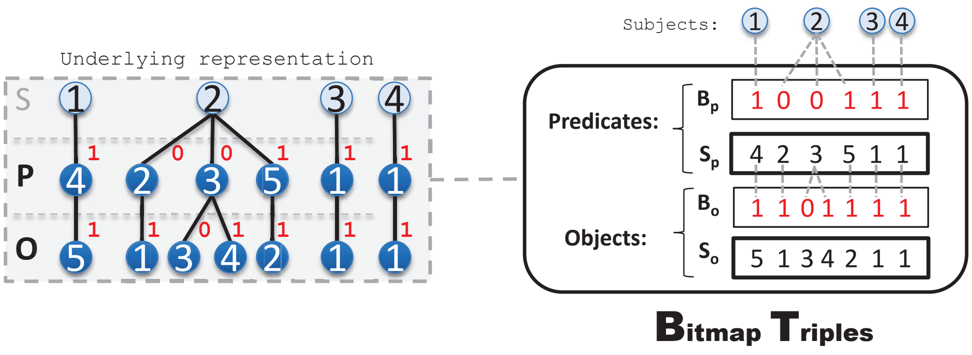


Figure 13.2: Detail of Bitmap Triples from Figure 13.1.

leaves as occurrences of the subject in the dataset). Each triple in the dataset is now represented as a full path root-to-leave in the corresponding tree.

The structure is encoded in the bitsequence and it can be interpreted as follows (Fernández et al., 2013). Let P_i be the list of predicates (*i.e. the predicate adjacency list*) for the i-th subject.

- The i-th 1-bit in $\mathcal{B}_p$ marks the end of P_i.
- The number of predicates in P_i can be obtained by subtracting the positions[1] of two consecutive 1-bit in $\mathcal{B}_p$.

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 second, third and fourth IDs in $\mathcal{S}_p$, hence $P_2 = \{2, 3, 5\}$.

In turn, object encoding is performed in a similar way, hence the interpretation follows an analogous approach. Let O_n be the list of objects (*i.e. the object adjacency list*) for the n-th *(subject,object)* pair.

- The n-th 1-bit in $\mathcal{B}_o$ marks the end of O_n.
- The number of objects in O_n can be obtained by subtracting the positions of two consecutive 1-bits in $\mathcal{B}_o$.

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 the ID 3. This predicate is related to the second subject as we have previously explained. Thus, this adjacency list stores all objects o related to the *(subject,predicate)* pairs $(2, 3)$.

13.1.2 BT Succinct Index

BT gives a practical representation of the graph structure which allows triples to be sequentially listed. However, direct accessing to the triples in the i-th list would require a sequential search until the i-th 1-bit is found in the bitsequence. Thus, we propose to exploit the basic concepts of succinct data structures presented in Chapter 2 (§2.4). In particular, we aim at building one succinct index per each $\mathcal{B}_p$ and $\mathcal{B}_o$ binary sequences, at consumption time.

As we have already introduced, there exist practical approaches which provides `rank`, `select` and `access` operations (described in §2.4) over the bitsequences in constant time, with a little spatial overhead. This overhead depends on the particular implementation but can be sized in $o(n)$, being n the original size of each bitsequence. In the following, we provide formalisms on how to access the encoded

[1]Note that we always consider that positions are numbered from "1", *i.e.* being a an array, $a[0]$ is the position number "1".

ID-graph through these primitives. We assume that BT keeps the original ordering by subject-predicate-object (SPO), but an analogous reasoning could be made for distinct orderings.

We denote $\mathcal{B}_p^*$ and $\mathcal{B}_o^*$ to the binary sequences with the succinct index already incorporated an loaded into memory at consumption time. The final configuration is then referred to as *BT**.

Definition 30 (BT*) *The Bitmap Triples configuration at consumption time, denoted* BT*, *is the set of succinct bitsequence indexes $\mathcal{B}_p^*$ and $\mathcal{B}_o^*$ together with the integer streams $\mathcal{S}_p$ and $\mathcal{S}_o$.*

We show below that this configuration provides efficient resolution for basic triple patterns. In particular, let G be an ID-triples graph, with $s \in S_G,\ p \in P_G,\ o \in O_G$ and v a SPARQL variable, $v \in V$, *BT** resolves:

- (s, p, o), which is equivalent to test the existence of a triple.
- (s, p, v), that is, retrieve all the objects for a given pair *(subject, predicate)*.
- (s, v, p), retrieving all predicates with which the given subject and object are related.
- (s, v, v), which means to retrieve all the information from a given subject.
- (v, v, v) is trivially achieved with an in-order scan.

As can be seen, all these triple pattern provide the subject and, from it, they navigate its corresponding tree. For explanation purposes, these triple patterns are given as terms, but it is clear that BT* manages an ID-graph. Thus, a pattern like (s, p, o) is equivalent to (i, j, k) after an ID replacement: $s = i$, $p = j$ and $o = k$.

Intuitively, the triple pattern resolution is based on using `select` operations to localize the adjacency list of predicates P_i for the i-th subject and the adjacency list of objects for the *(i,j)* pair. We first provide an example of resolution, and next we generalize the process.

Example. Let us illustrate the resolution of checking the existence of the triple $(2, 3, 4)$ in our running example (Figure 13.2). First, one has to locate the adjacency list of predicates for the second subject: P_2. This is equivalent to locate the initial ($beginP$) and final ($endP$) positions of the list. Note that the 2-nd `1`-bit in $\mathcal{B}_p^*$ marks the end of P_2. Thus, the final position is achieved with a `select` operation over $\mathcal{B}_p^*$:

$endP = select_1(\mathcal{B}_p^*, 2) = 4$

In turn, as every list begins at the end of the previous list (or zero if this is the first list), $beginP$ is:

$beginP = select_1(\mathcal{B}_p^*, 1) + 1 = 1 + 1 = 2$

Then, P_2 is retrieved from $\mathcal{S}_p[beginP, endP]$, which returns $\mathcal{S}_p[2, 4] = \{2, 3, 5\}$. In this list of predicates, we look for the predicate 3, as we are checking the existence of the triple $(2, 3, 4)$. This search can be performed on a binary search, which actually returns $position = 2$ as the predicate 3 actually is in the second position. Thus, we are positioned on the third *(subject,predicate)* pair, $\mathcal{S}_p[3] = 3$. Note that we calculate that this is the third pair with $beginP + position - 1$ = 2+2-1 = 3.

Next, the third list of objects, O_3 is marked in the third `1`-bit in $\mathcal{B}_o^*$, then we retrieve their delimiting positions as:

$endO = select_1(\mathcal{B}_o^*, 3) = 4$
$beginO = select_1(\mathcal{B}_o^*, 2) + 1 = 2 + 1 = 3$

Therefore, the list O_3 is retrieved as $\mathcal{S}_o[beginO, endO]$, which returns $\mathcal{S}_o[3, 4] = \{3, 4\}$. Again, we perform on it a binary search to look for the object 4 as we are checking the existence of the triple $(2, 3, 4)$. It actually exists, an therefore the final result is *true*. $\square$

Algorithm 1 `findPredicate(i)`

```
function FINDPREDICATE(i)
    $endP \leftarrow \mathbf{select_1}(\mathcal{B}_p^*, i)$;
    $beginP \leftarrow \mathbf{select_1}(\mathcal{B}_p^*, i-1)+1$;
    $size_{P_i} \leftarrow endP - beginP$;
    $P_i \leftarrow \mathcal{S}_p[beginP, endP]$;
    return $(P_i, beginP)$;
end function
```

Algorithm 2 `findObject(x)`

```
function FINDOBJECT(x)
    $endO \leftarrow \mathbf{select_1}(\mathcal{B}_o^*, x)$;
    $beginO \leftarrow \mathbf{select_1}(\mathcal{B}_o^*, x-1)+1$;
    $size_{O_x} \leftarrow endO - beginO$;
    $O_n \leftarrow \mathcal{S}_o[beginO, endO]$;
    return $O_x$;
end function
```

The previous example shows how to test the existence of a triple by means of i) `select` operations[2] over $\mathcal{B}_p^*$ and $\mathcal{B}_o^*$, ii) access to given positions in $\mathcal{S}_p$ and $\mathcal{S}_o$, and iii) binary searches over the intermediate retrieved adjacency lists. In fact, the rest of the presented triple patterns can be resolved in a similar way. For instance, if no object is given, such as $(2, 3, v)$, the process runs exactly the same until the last step, in which all the list of valid objects, O_3, is returned. In turn, a pattern $(2, v, 4)$ starts in the same way, but it repeats the last step for every predicate in the adjacency list of predicates.

We generalize this process and distinguish below four primitives which are sequentially performed to test a ID-triple existence and therefore they constitutes the basis to resolve the aforementioned triple patterns in BT^*.

- `findPredicate`$(i) \rightarrow (P_i, beginP)$. This function returns the list of predicates related to the subject `i`, P_i, and the position $beginP$ in which this list begins in $\mathcal{S}_p$. For instance, as we showed in our running example, $findPredicate(2) = (\{2, 3, 5\}, 2)$.

  Algorithm 1 generalizes the required operations that we have illustrated in the example. First, we obtain the delimiting positions of P_i (Lines 2-3). The size of the list is also calculated for future estimation purposes (Line 4). Then, we retrieve the list of predicate IDs from $\mathcal{S}_p$ (Line 5) and return the result (Line 6).

- `filterPredicate`$(P_i, j) \rightarrow position$. It performs a binary search on P_i and returns the position of the predicate j in P_i, or 0 if it is not in the list. For instance, $filterPredicate(P_2, 3) = 2$ in our running example, as the predicate 3 is located in the second position of $\{2, 3, 5\}$.

  Note that the predicate j is located then in $\mathcal{S}_p[n]$ where $n = beginP + position - 1$. In other words, the objects for the (i, j) pair is represented in the n-th list in $\mathcal{S}_o$.

- `findObject`$(x) \rightarrow O_x$. This function returns the x-th list of objects, O_x, which is related to the x-th *(subject,predicate)* pair represented in $\mathcal{S}_p[x]$. For instance, as shown in our running example, $findObject(3) = \{3, 4\}$.

  Algorithm 2 generalizes the required operations. First, we obtain the delimiting positions of O_x (Lines 2-3). Similarly to the previous case, the size of the list is also calculated for future estimation purposes (Line 4). Finally, the list of objects is retrieved from $\mathcal{S}_o$ (Line 5) and returned (Line 6).

- `filterObject`$(O_n, k) \rightarrow position$: performs a binary search on O_j and returns the position of the object k in O_j, or 0 if it is not in the list. In our example, $filterObject(\{3, 4\}, 4) = 2$.

[2]Note that no `rank` operations are used, though a `select` operation can be performed which successive `rank` operations

Algorithm 3 `(i,j,k)` TP resolution

```
1: $(P_i, beginP) \leftarrow$ FINDPREDICATE$(i)$;
2: $position \leftarrow$ FILTERPREDICATE$(P_i, j)$;
3: $n \leftarrow beginP + position - 1$;
4: $O_n \leftarrow$ FINDOBJECT$(n)$;
5: $pos \leftarrow$ FILTEROBJECT$(O_n, k)$;
6: if $(pos \neq 0)$ then
7:     output$(true)$;
8: else
9:     output$(false)$;
10: end if
```

Algorithm 4 `(i,j,v)` TP resolution

```
1: $(P_i, beginP) \leftarrow$ FINDPREDICATE$(i)$;
2: $position \leftarrow$ FILTERPREDICATE$(P_i, j)$;
3: $n \leftarrow beginP + position - 1$;
4: $O_n \leftarrow$ FINDOBJECT$(n)$;
5: output$(O_n)$;
```

The resolution of the triple patterns (TP) by means of these four primitives is performed as follows:

- (i, j, k) - Algorithm 3: As shown in the previous example, the four primitives are called in order (Lines 1-5). The last operation, $filterObject$ returns the position of the given object in its adjacency list, if present. A value of 0 stands for the nonexistence of the triple pattern and then the output is false (Line 9). Otherwise the output is true (Line 7).
- (i, j, v) - Algorithm 4: The firsts three function calls (Lines 1-4) are exactly equal to the previous case. The last step does not check an object, but it directly outputs all the adjacency list of objects (Line 5).
- (i, v, k) - Algorithm 5: First, the process runs similar to the previous cases, obtaining the list of predicates for the given subject, O_i (Line 1). Then, for each predicate (Line 2), it retrieves the corresponding list of objects (Line 4) and tests the existence of the k object (Line 5). A nonzero value stands for the existence of the object and thus the predicate is outputted (Lines 5-6). We assume here a stream of output values matching the predicate variable.
- (i, v, v) - Algorithm 6: First, we retrieve the list of predicates for the given subject, O_i (Line 1). Then, for each predicate in the list (Line 2), we retrieve the corresponding list of objects (Lines 3-4). We assume here a stream of pairs as output values (Line 5), composed of the matching predicate and the list of objects for this predicate.
- (v, v, v) - Algorithm 7: This pattern retrieves all the dataset in order, and it is performed with a double loop. For every subject[3] (Line 1), we perform as the previous pattern, retrieving the list of predicates (Line 2) and, for each predicate (Line 3), the corresponding list of objects (Lines 4-5). In this case, we assume to output a stream of three elements composed of the subject, predicate, and list of objects for this *(subject,predicate)* pair (Line 6).

Algorithmic costs. The aforementioned metrics enables an accurate estimation of the TP resolution in Bitmap Triples. Table 13.1 summarizes the cost of each TP and the average cost in a general case. Agreeing an O(1) cost for `select` operations, and logarithmic costs of binary searches, the explanation of the algorithmic cost is straightforward. First, let us detail the cost of the four primitives:

- $findPredicate(i)$: It performs `select operations`, O(1), and basic retrieval on an array, O(1). Thus, this operation runs in time O(1).

[3]We assume that we know the maximum subject ID, which is always possible given a basic mapping.

Algorithm 5 `(i,v,k)` TP resolution

```
1: $(P_i, beginP) \leftarrow$ FINDPREDICATE$(i)$;
2: for (it:=0 to $P_i.size()$); do
3:     $n \leftarrow beginP + it$;
4:     $O_n \leftarrow$ FINDOBJECT$(n)$;
5:     if (FILTEROBJECT$(O_n, k) \neq 0$) then
6:         output$(P_i[it])$;
7:     end if
8: end for
```

Algorithm 6 `(i,v,v)` TP resolution

```
1: $(P_i, beginP) \leftarrow$ FINDPREDICATE$(i)$;
2: for (it:=0 to $P_i.size()$); do
3:     $n \leftarrow beginP + it$;
4:     $O_n \leftarrow$ FINDOBJECT$(n)$;
5:     output$(P_i[it], O_n)$;
6: end for
```

Algorithm 7 `(v,v,v)` TP resolution

```
1: for (i:=0 to $maxSubjectID$); do
2:     $(P_i, beginP) \leftarrow$ FINDPREDICATE$(i)$;
3:     for (it:=0 to $P_i.size()$); do
4:         $n \leftarrow beginP + it$;
5:         $O_n \leftarrow$ FINDOBJECT$(n)$;
6:         output$(i, P_i[it], O_n)$;
7:     end for
8: end for
```

- $filterPredicate(P_i, j)$: It performs a binary search on the list of predicates related to the subject i. As stated, the size of this list is delimited by the labeled degree $degL^{-}(i)$, and thus it runs in time O($log(degL^{-}(i))$). In the general case, we take into account the mean value of all predicates, hence this operation runs in an average time O($log(\overline{degL^{-}}(G))$)[4].

  Remember that, in all the considered datasets, the value of the labeled degree was less than 20 (even in the biggest datasets).

- $findObject(n)$: It performs similar to $findPredicate$, with `select` and basic operations, running in time O(1).

- $filterObject(O_n, k)$: It performs similar to $filterPredicate$, but with a binary search on the list of objects related to the given $(subject, predicate)$ pair. The size of this list for (i, j) is exactly delimited by the partial degree $deg^{--}(i, j)$, hence this operation runs in O($log(deg^{--}(i, j))$). Remember that, in all the evaluated datasets, this partial degree was close to 1. In such case, a binary search is even unnecessary (there is only one element in the list). In any case, the general case runs in an average time O($log(\overline{deg^{--}}(G))$).

As shown, all these operations runs efficiently and they are well delimited by the presented metrics. Thus, TP resolution costs can be summarized as follows:

- (i, j, k) - Algorithm 3: As the four primitives are called in order (Lines 1-5), this TP runs in a logarithmic time with respect to the size of the involved predicate list and object list. Formally, it runs in time O($log(degL^{-}(i)) + log(deg^{--}(i, j))$). For the general case, this TP runs in an average time O($log(\overline{degL^{-}}(G)) + log(\overline{deg^{--}}(G))$).

[4]Note that we always consider hereinafter that the logarithm of an average is an upper limit due to the Jensen's inequality (Kuczma, 2008). In short, for a concave function f and numbers $x_1, x_2, \cdots, x_n$ in its domain, it is true that $f(\frac{\sum x_i}{n}) \geq \frac{\sum f(x_i)}{n}$. Thus, it remains true in our case when $f(x) = log(x)$, being x one of the degrees in our metrics.

Triple Pattern	Average time
(i, j, k)	O($log(\overline{degL^-}(G)) + log(\overline{deg^{--}}(G))$)
(i, j, v)	O($log(\overline{degL^-}(G))$)
(i, v, k)	O($\overline{degL^-}(G) * log(\overline{deg^{--}}(G))$)
(i, v, v)	O($\overline{degL^-}(G)$)
(v, v, v)	O($\|S\| * \overline{degL^-}(G)$)

Table 13.1: Triple pattern resolution times on BT*.

- (i, j, v) - Algorithm 4: It runs similar to the previous case, but it obviates the last binary search on objects (the operation $filterObject$) as it returns all the list. Thus, it runs in time O($log(degL^-(i))$). For the general case, this TP runs in an average time O($log(\overline{degL^-}(G))$).
- (i, v, k) - Algorithm 5: As shown, for each predicate related to the subject i, it retrieves the object list and check the existence of the given object. As this number of predicates is the labeled degree $degL^-(i)$, it is clear that it performs in time O($degL^-(i) * log(deg^{--}(i))$), where $deg^{--}(i)$ is the maximum partial degree of all pairs (i, v). The average time is then O($\overline{degL^-}(G) * log(\overline{deg^{--}}(G))$).
- (i, v, v) - Algorithm 6: The resolution performs similar to the previous case, but it obviates the last binary search on objects (the operation $filterObject$) as it returns all the list. Thus, the resolution of this TP runs in time O($degL^-(i)$). In other words, the algorithm performs $degL^-(i)$ iterations, and the cost of each iteration is in O(1). In the general case, the average time is then O($\overline{degL^-}(G)$).
- (v, v, v) - Algorithm 7: This pattern retrieves all the dataset in order with a double loop. One can see that the cost of each iterations run in time O(1) (find operations). Thus, it performs in an average time O($|S| * \overline{degL^-}(G)$).

13.1.3 Application

HDT was originally intended for publication and exchange but, as shown above, its Bitmap Triples component provides enough information for efficient RDF retrieval once loaded. The so-called Bitmap Triples configuration at consumption time (BT*), provides an SP-O index which allows some triple patterns to be efficiently resolved (Table 13.1). In fact, one could argue that these TP cover a vast range of real SPARQL queries. As we stated in an empirical study of real-world SPARQL queries (Arias et al., 2011), most SPARQL queries contains just one simple triple pattern. The proportion of such simple queries reaches up to 66% in *Dbpedia* and 97% in the *Semantic Web Dog Food* logs. If we analyze the TP combinations used in all queries (including those from BGPs), the combination of patterns resolved by BT* cover the 89% of the TP combinations in the Dbpedia query logs, and the 50% of those from SWDF.

If we combine these results we can state that, in plain words, the exchanged HDT Bitmap Triples, after a lightweight loading process at consumption time (resulting in the so-called BT*) can resolve about the 50% of the most common queries in SPARQL.

Note that, in some scenarios, the resolved TP could cover all the requirements. For instance, that could be the case of applications a) checking triple existence, b) making simple restrictions over subjects or b) traversing all the graph.

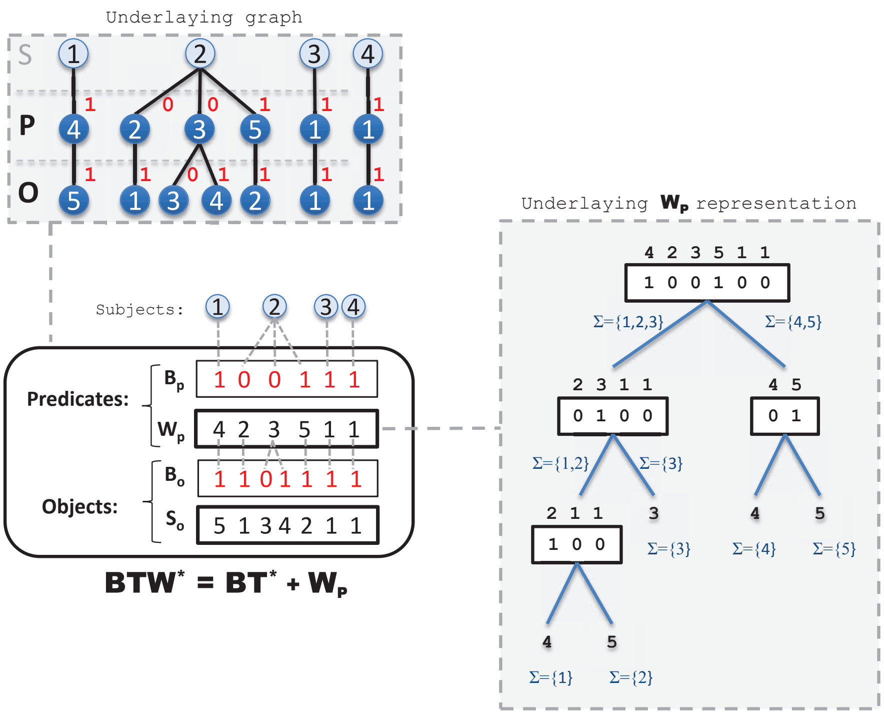


Figure 13.3: BTW*: The proposed encoding of Bitmap Triples with a Wavelet Tree W_p in predicates.

13.2 Additional Compressed Succinct Data Structures

The original Bitmap Triples (BT) representation draws adjacency lists prioritized by subject. This decision addresses fast querying for the patterns providing the subject, as presented above, but makes retrieval by predicate and object difficult.

This section presents how HDT can be enhanced with additional indexes at consumption time in order to resolve all kind of triple pattern in SPARQL. In particular, we propose a Wavelet Tree-based solution for PS-O indexing (§13.2.1) and an additional adjacency list for OP-S indexing (§13.2.2).

13.2.1 A Wavelet Tree-based Solution for PS-O Indexing

In this section we focus on enabling access by predicate on top of BT*. That is, we address the resolution of the Triple Patterns presented below:

- (v, p, v), which means to retrieve all the information from a given predicate.
- (v, p, o), that is, retrieve all the subjects for a given pair *(predicate, object)*.

In both cases, the occurrences of each predicate must be quickly located and this operation demands a direct access to the predicate stream S_p. However, as predicates are scattered along the stream, locating all predicate occurrences in BT* demands a full scanning of the sequence, resulting in poor performance.

Thus, the predicate-based retrieval demands indexed access to $\mathcal{S}_p$ at consumption time, which could be satisfied with multiple alternatives. For instance, an additional B+ index could be built once BT* is loaded. Nevertheless, in order to keep the same compact conception of the representation, one should consider that $\mathcal{S}_p$ can be seen as a general sequence of symbols and then succinct indexes providing efficient rank/select and access operations can be build on top of this sequence.

In particular, we propose the consumer to load $\mathcal{S}_p$ on a Wavelet Tree structure (see the definition and basic concepts in Section 2.4.2). Whereas S_p lists plain predicates, the Wavelet Tree, $\mathcal{W}_p$, represents

Algorithm 8 `occsPred(j)`

```
1: function OCCSPRED(j)
2:     $numOccs \leftarrow \mathbf{rank_j}(\mathcal{W}_p, \mathcal{W}_p.size());$
3:     for ($x = 1$ to $numOccs$); do
4:         $posPred[\,] \leftarrow \mathbf{select_j}(\mathcal{W}_p, x);$
5:     end for
        return $posPred$;
6: end function
```

them by a balanced tree of height $h = \lceil log|P| \rceil$. Figure 13.3 shows the schema of the representation for the example in Figure 13.2. The configuration is then referred to as *BTW**.

Definition 31 (BTW*) *The Bitmap Triples configuration at consumption time enhanced with a Wavelet Tree index, denoted* BTW**, is the set of succinct bitsequence indexes $\mathcal{B}_p^*$ and $\mathcal{B}_o^*$, the succinct Wavelet Tree $\mathcal{W}_p$ together with the integer stream $\mathcal{S}_o$.*

In the following, let us treat $\mathcal{W}_p$ as a black box holding the predicates and serving the aforementioned operations (described in detail in §2.4.2):

- `rank`$_j(\mathcal{W}_p, m)$ counts the occurrences of the predicate j in $\mathcal{W}_p[1, m]$.
- `select`$_j(\mathcal{W}_p, m)$ locates the position for the m-th occurrence of the predicate j in $\mathcal{W}_p$.
- `access`$(\mathcal{W}_p, m)$ returns the symbol in $\mathcal{W}_p[m]$.

Note that the W_P structure adds an additional overhead of $o(n)log|P|$ bits to the space used in the original $\mathcal{S}_p$, and serves all these operations in time $O(log|P|)$ (see §2.4.2). This is an acceptable cost for our purposes because of the small number of predicates used, in practice, for RDF modeling (see §4.2).

The Wavelet Tree structure allows access by predicate to be supported on one new primitive retrieving the position of each predicate occurrence in the subject adjacency lists:

- `occsPred(j)`: returns the positions of the predicate j in $\mathcal{W}_p$. This operation is described in Algorithm 8. First, a simple `rank` (Line 2) counts the number of occurrences of the predicate j along the full size of $\mathcal{W}_p$. Then, for each occurrence of j (Line 3), it makes use of a `select` operation to get the position of the occurrence (Line 4), storing an array of positions which is finally returned as result.

  For instance, in the example in Figure 13.3, the operation `occPred(1)` runs as follows. First, it counts the number of total occurrences of the predicate 1, which are actually 2. Then, we iterate obtaining each position: the operations $select_1(W_p, 1)$ and $select_1(W_p, 2)$ obtain the position 5 and 6 respectively which are returned in an array as result.

Thus, the resolution of the triple patterns by predicate is performed as follows:

- (v, j, v) - Algorithm 9: First, the process obtains the list of occurrence positions of the given predicate j, $posPred$ (Line 1). Then, for each occurrence (Line 2), it retrieves the corresponding subject (Line 3) and list of objects (Line 4), which are outputted as result (Line 5). Note that it is simple to obtain the related subject of a predicate position, as it is marked with the number of

Algorithm 9 `(v,j,v)` TP resolution

```
1: $posPred[\,] \leftarrow$ OCCSPRED$(j)$;
2: for $(x = 1$ to $posPred.size())$; do
3:     $subject \leftarrow rank_1(B_p^*, posPred[x] - 1) + 1$;
4:     $O_n \leftarrow$ FINDOBJECT$(posPred[x])$;
5:     $\mathbf{output}(subject, O_n)$;
6: end for
```

Algorithm 10 `(v,j,k)` TP resolution

```
1: $posPred[\,] \leftarrow$ OCCSPRED$(j)$;
2: for $(x = 1$ to $posPred.size())$; do
3:     $O_n \leftarrow$ FINDOBJECT$(x)$;
4:     $posObject \leftarrow$ FILTEROBJECT$(O_n, k)$;
5:     if $(posObject \neq 0)$ then
6:         $subject \leftarrow rank_1(B_p^*, posPred[x] - 1) + 1$;
7:         $\mathbf{output}(subject)$;
8:     end if
9: end for
```

1-bits in B_p up to the previous position plus one[5]. This is simply retrieved with a `rank` operation over the B_p^* component[6] (Line 3).

For instance, let us explain the resolution of the pattern $(v, 1, v)$ in the example in Figure 13.3. The process first uses the Wavelet Tree operation $occsPred(1)$ to retrieve the predicate positions 5 and 6 in which the predicate 1 occurs. Next, it iterates over these positions. For position 5, $rank_1(B_p^*, 4) + 1$ returns 3, which means that the subject ID 3 is related with this position. The object list for position 5 is retrieved by $findObject(5) = \{1\}$. The first outputted solution is then $(3, 1)$, *i.e.*, subject=3 and object=1. The process is similar for position 6, obtaining the result $(4, 1)$.

- (v, j, k) - Algorithm 10: It performs similar to the previous case, but it restricts to those objects equal to the given object k. First, the process obtains the list of occurrence positions of the given predicate (Line 1). Then, for each occurrence (Line 2), it retrieves the list of objects (Line 3), and tests the existence of the k object (Line 4). A nonzero value stands for the existence of the object and thus the subject is retrieved similarly to the previous case (Line 5) and outputted as a valid result (Line 7).

  For instance, let us briefly present the resolution of the pattern $(v, 3, 4)$ in the example in Figure 13.3. The process starts making use of $occsPred(3)$ to retrieve the position in which the predicate ID 3 takes place in the Wavelet Tree. In this case, the retrieved position is 3. Next, we obtain the list of objects related to the third subject-predicate pair with $O_3 = findObject(3) = \{3, 4\}$. Then, we test if the object 4 is in such list with $filterObject(O_3, 4)$. The object actually exists (in position 2), and the related ID of the subject is obtained, as stated, with $rank_1(B_p^*, 2) + 1 = 2$. Thus, the outputted value is 2 stating that the subject ID 2 is solution for this pattern.

[5]It is easy to see that this formula allows to discount the intermediate zeros denoting repetitions.

[6]We assume here that $rank_1(0) = 0$.

Algorithmic costs. The Wavelet Tree contributes with a `PS-O` index which allows the TP by predicate to be efficiently resolved. Nevertheless, it is worth noting that the Wavelet Tree runs in time $O(log|P|)$ for all `rank`, `select` and `access` operations. As shown in the TP resolution on BT^* (§13.1.2), the sequence of predicates S_p is always accessed. In particular, it is easy to see that it is accessed by the $findPredicate(i)$ function (Algorithm 1, Line 5) which is called by all TP resolution algorithms (Algorithms 3 to 7, Line 1). Whereas this access was previously performed in BT* in time O(1) (we have assumed the sequence has been loaded into an array), in BTW* the substitution of S_p by the Wavelet Tree W_p makes this time $O(log|P|)$.

Table 13.2 updates this overhead of time for the previous patterns working on BTW*. Note that for the special case of (v, v, v) this overhead is a multiplicative factor as we iterate over the number of subjects. Nevertheless, we have previously justified the reduced time overhead in the limited number of different predicates per dataset (see §4.2).

The latest two rows on Table 13.2 show the estimation of time for the novel patterns (v, j, o) and (v, j, v) which can now be resolved thanks to the Wavelet Tree. The explanation of these costs is also simple. We first detail the cost of the $occsPred(j)$ primitive:

- $occsPred(j)$: It performs a `rank` operation over the Wavelet Tree, $O(log|P|)$ and for each occurrence it retrieves its position with a `select` operation, $O(log|P|)$. The number of occurrences of a predicate in the stream is perfectly describe by its "predicate in-degree" $deg^+_P(j)$, hence this primitive runs in time O($log|P| + (log|P| * deg^+_P(j))$), that is, time O($log|P| * (deg^+_P(j) + 1)$). The general case runs in an average time O($log|P| * (\overline{deg^+_P}(G) + 1)$).

As expected, the cardinality of each predicate has a strong influence in the efficient performance of the primitive, and consequently of the TP resolutions. The cost of these resolution can be summarized as follows:

- (v, j, v) - Algorithm 9: This algorithm first calls the $ocssPred(j)$ primitive (Line 1), O($log|P| * (deg_P(j) + 1)$). Next, for each retrieved position, it uses a `rank` operation over the bitmaps, O(1), and calls a $findObject$ primitive, O(1). We can assume an efficient implementation which obviates the loop over the positions (Line 2) as it can be done directly as soon as we get the positions in the $occPreds$ code (line 4). Thus, formally, it runs in time O($log|P| * (deg^+_P(j) + 1)$). For the general case, this TP runs in an average time O($log|P| * (\overline{deg^+_P}(G) + 1)$).

- (v, j, k) - Algorithm 10: The algorithm performs similar to the previous case. It first calls the $ocssPred(j)$ primitive (Line 1), O($log|P| * (deg^+_P(j) + 1)$). Next, for each retrieved position, it calls a $findObject$ primitive, O(1), and a $filterObject$ primitive, O($log(deg^{--}(x, j)$) being x the subject involved in each case. Finally, whenever the object k is found, it uses a `rank` operation over the bitmaps, O(1). Assuming again an efficient implementation obviating the loop (Line 2) as part of $occPreds$ code (line 4), the general case runs in an average time O($log|P| * (\overline{deg^+_P}(G) + 1) + log(\overline{deg^{--}}(G))$). Note that the latest component, $log(\overline{deg^{--}}(G))$, computes all the $filterObject$ calls. In practice, this mean partial out-degree is close to 1 in our evaluated datasets (see §4.3.4).

Application. We have shown that a Wavelet Tree can effectively replace the predicate stream at consumption time, conforming the so-called BTW* configuration. The integrated `PS-O` index provides access by predicate, allowing efficient resolution of two novel TP (latest two rows in Table 13.2).

It is worth mentioning that these novel possibilities are, as stated, at the cost of a slight $O(log|P|)$ overhead in time for the rest of patterns and an extra space of $o(n)log|P|$ bits to the space used in the original $\mathcal{S}_p$.

Triple Pattern	Average time
(i,j,k)	$O(log\|P\| + log(\overline{degL^{-}}(G)) + log(\overline{deg^{--}}(G)))$
(i,j,v)	$O(log\|P\| + log(\overline{degL^{-}}(G)))$
(i,v,k)	$O(log\|P\| + \overline{degL^{-}}(G) * log(\overline{deg^{--}}(G)))$
(i,v,v)	$O(log\|P\| + \overline{degL^{-}}(G))$
(v,v,v)	$O(\|S\| * log\|P\| * \overline{degL^{-}}(G))$
(v,j,v)	$O(log\|P\| * (\overline{deg_P^{+}}(G) + 1))$
(v,j,o)	$O(log\|P\| * (\overline{deg_P^{+}}(G) + 1) + log(\overline{deg^{--}}(G)))$

Table 13.2: Triple pattern resolution times on BTW*.

Thus, consumer applications should consider this configuration over previous BT* in scenarios in which a) Triple patterns by predicate (v,j,v) or (v,j,k) are required or b) efficient access by predicate is required. Nevertheless, the study on SPARQL query logs by Arias et al. (2011) shows that accesses by predicate are less common for the first pattern (v,j,v) than for (v,j,k). For instance, 3.45% of the TPs in *Dbpedia* are of type (v,j,v), versus 7% of (v,j,k). In SWDF, up to 4.21% are of type (v,j,v), versus a significant 46.08% of (v,j,k).

These results shows that, in some scenarios such as the one pointed in this study, the complete BTW* can resolve about the 99% of those queries with one simple TP. Averaging over the total types of queries (including BGPs), we can state that BTW* can cover the 80% of the total queries asked to a dataset.

13.2.2 An Additional Adjacency List for `OP-S` Indexing

The Wavelet-Tree based enhancement in BTW* leaves object-based access as the only non-efficient retrieval in our approach. As we illustrated in Figures 13.2 and 13.3 (for BT* and BTW* respectively), objects are always represented as leaves of the tree drawn for each adjacency list. Thus, all the occurrence of an object are scattered throughout the sequence S_o which prevent this from efficient access by object.

In this case, we require an additional index `OP-S` which allows adjacency lists to be traversed from the leaves in an efficient manner and supports the following not addressed TP:

- (v,v,o), retrieving all the information from a given object.

In addition, the `OP-S` index would also help in efficient resolution of the TP (v,i,k). Although this TP was addressed by means of the Wavelet Tree W_p in the BTW* configuration, its resolution was not straightforward. In fact, the previous approach run in time proportional to the product of the logarithm of the different number of predicates and the number of triples in which the predicate i is present (see Table 13.2). Note that W_p contributes with a `PS-O` index, whereas the TP (v,i,k) would benefit from a more appropriated `OP-S` index.

One could be tempted to address this `OP-S` index with an structure like another Wavelet Tree substituting the sequence $\mathcal{S}_o$. However, this would become highly inefficient: the operations in the Wavelet Tree run in time proportional to its height, which is the logarithm of the involved vocabulary. In this case, the vocabulary consists of all different objects of a datasets, and it can be massively big (see experimental results in Section 4.2). Thus, it would result in very expensive operations.

We propose to replicate instead an adjacency list of objects occurrences, referred to as *O-Index*. This representation is illustrated in the example in Figure 13.4. The *O-Index* draws an adjacency list with one list per different object. Each list stores the positions in S_o in which this object appears. In other words, each list clusters all the occurrences of objects, each one related to a $(subject, predicate)$ pair.

As in BT, the underlying adjacency list of *O-Index* implicitly represents the objects and makes use of an integer sequence and a bitsequence: $\mathcal{S}_{oP}$ stores the list of positions in S_o for each object, whereas the bitsequence $\mathcal{B}_{oP}$ is used for representing the cardinalities of the lists as in the upper levels. For instance,

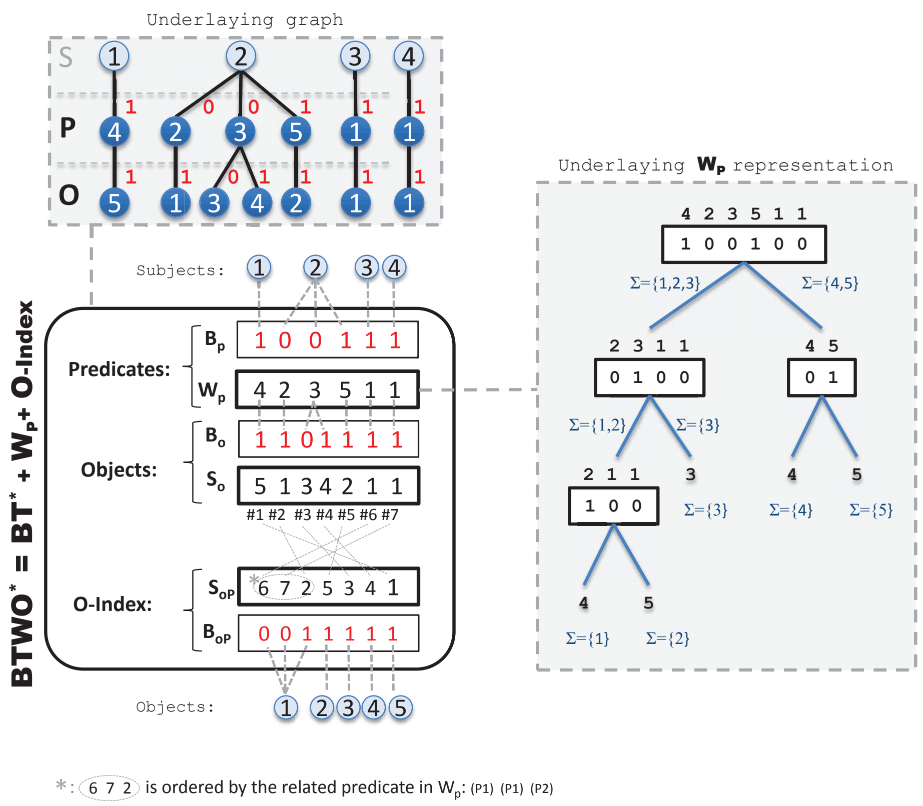


Figure 13.4: BTWO*: Bitmap Triples encoding enhanced with two additional indexes by predicate (W_p) and object (*O-Index*).

in Figure 13.4, the first 1-bit in $\mathcal{B}_{oP}$ delimits the end of the list for the first object. This list stores the values {6,7,2}, which, as can be seen, are the positions in S_o in which the first object occurs.

The only difference between a common adjacency list is that we sort the positions in ascending order of the related predicate. That is, in the previous example, the positions {6,7,2} corresponds to related predicates {1,1,2}. As we will explain below, this does not affect the rest of operations and it allows for query performance optimizations.

The complete configuration in Figure 13.4 is referred to as *BTWO**.

Definition 32 (BTWO*) *The Bitmap Triples configuration at consumption time enhanced with a Wavelet Tree index and a* O-Index*, denoted* BTWO**, is the set of succinct bitsequence indexes* $\mathcal{B}_p^*$, $\mathcal{B}_o^*$ *and* $\mathcal{B}_{oP}^*$*, the succinct Wavelet Tree* $\mathcal{W}_p$*, and the integer streams* $\mathcal{S}_o$ *and* $\mathcal{S}_{oP}$.

In relative terms, this *O-Index* has a significant impact in the total configuration of BTWO*. In particular, note that $\mathcal{S}_{oP}$ and $\mathcal{B}_{oP}$ are of size n (being n the total number of triples) as they hold one element per occurrence. $\mathcal{B}_{oP}$ stores bits and the $o(n)$ overhead for the operations on the bitsequences, whereas $\mathcal{S}_{oP}$ is an array of positions, each of them represented with $\lceil log|n|\rceil$ bits. Thus, it is clear that the *O-Index* adds an overhead of $n\lceil log|n|\rceil + o(n)$ bits. However, in absolute terms, the total size required by BTWO is small in comparison to that required by the other competitive solutions in the state of the art (as we will see in Section 13.3).

This index OP-S enables efficient object-based retrieval trough one new primitive which traverses adjacency lists from the leaves:

- occsObj(k): returns the positions of the object k in $\mathcal{S}_p$. This operation is described in Algorithm 11. First, two select operations on $\mathcal{B}_{oP}$ delimit the list of positions in $\mathcal{S}_o$ for the given object

Algorithm 11 `occsObj(k)`

```
1: function OCCSOBJ(k)
2:     $endO \leftarrow \mathbf{select_1}(\mathcal{B}^*_{oP}, k)$;
3:     $beginO \leftarrow \mathbf{select_1}(\mathcal{B}^*_{oP}, k-1)+1$;
4:     for ($x = beginO$ to $endO$); do
5:         $posObj[\,] \leftarrow \mathbf{rank_1}(\mathcal{B}^*_o, \mathcal{S}^*_{oP}[x]-1)+1$;
6:     end for
       return $posObj$;
7: end function
```

Algorithm 12 `(v,v,k)` TP resolution

```
1: $posObj[\,] \leftarrow$ OCCSOBJ$(k)$;
2: for ($x = 1$ to $posObj.size()$); do
3:     $predicate \leftarrow access(W^*_p, posObj[x])$;
4:     $subject \leftarrow rank_1(B^*_p, posObj[x]-1)+1$;
5:     $\mathbf{output}(subject, predicate)$;
6: end for
```

Algorithm 13 `(v,j,k)` TP resolution in BTWO*

```
1: $posObj[\,] \leftarrow$ OCCSOBJ$(k)$;
2: for ($x = 1$ to $posObj.size()$); do
3:     $predicate \leftarrow access(W^*_p, posObj[x])$;
4:     if $(predicate = j)$ then
5:         $subject \leftarrow rank_1(B^*_p, posObj[x]-1)+1$;
6:         $\mathbf{output}(subject)$;
7:     end if
8: end for
```

(Line 2-3). Note that the position is $\mathcal{S}_p$ of a position in $\mathcal{S}_o$ can be obtained the counting the number of 1-bits in $\mathcal{B}_p$ up to the previous position plus one (in order to discount intermediate zero-values). Thus, for each occurrence of k (Line 4), it makes use of a `rank` operation on $\mathcal{B}_o$ to get the position of the occurrence in $\mathcal{S}_p$ (Line 5), storing an array of positions which is finally returned as result.

For instance, in the example in Figure 13.4, the operation `occsObj(1)` runs as follows. First, the `select` operations delimit the list of positions for the object 1, being this list `{6,7,2}`. We iterate obtaining each position: the operations $rank_1(B_o, 5)+1$, $rank_1(B_o, 6)+1$ and $rank_1(B_o, 1)+1$ returns 5, 6 and 2 respectively. These are the positions in S_p of those predicates related with the object 1, which are returned in an array as result.

The resolution of triple patterns by object is performed as follows:

- (v, v, k) - Algorithm 12: First, the process obtains the positions of the object k in $\mathcal{S}_p$, $posObj$ (Line 1). Then, for each occurrence (Line 2), it retrieves the corresponding predicate (Line 3) and subject (Line 4), which are outputted as result (Line 5). Note that the related predicate is achieved directly accessing each position in the Wavelet Tree. In turn, as previously stated, the related subject of a predicate position is simply retrieved with a `rank` operation over the B^*_p component.

  For instance, let us explain the resolution of the pattern $(v, v, 1)$ in the example in Figure 13.4. The process first uses the *O-Index* operation $occsObj(1)$ to retrieve the S_o positions 5, 6 and 2 in which the object 1 occurs. Next, it iterates over these positions. For position 5, $access(W^*_p, 5)$ returns 1, which means that the predicate ID 1 is related with this position. In turn, the related subject is retrieved with $rank_1(B^*_p, 4)+1 = 3$. Then, the first outputted solution is $(3, 1)$, that is, subject=3 and predicate=1. The process is similar for position 6 and 2, obtaining the results $(4, 1)$ and $(2, 2)$ respectively.

- (v, j, k) - Algorithm 13: The process runs almost similar to the previous case. The only difference is that it does not retrieve all the subjects, as it previously check if the related predicate is equal to

Triple Pattern	Average time
(i, j, k)	O$(log\vert P\vert + log(\overline{degL^-}(G)) + log(\overline{deg^{--}}(G)))$
(i, j, v)	O$(log\vert P\vert + log(\overline{degL^-}(G)))$
(i, v, k)	O$(log\vert P\vert + \overline{degL^-}(G) * log(\overline{deg^{--}}(G)))$
(i, v, v)	O$(log\vert P\vert + \overline{degL^-}(G))$
(v, v, v)	O$(\vert S\vert * log\vert P\vert * \overline{degL^-}(G))$
(v, j, v)	O$(log\vert P\vert * (\overline{deg^+_P}(G) + 1))$
(v, v, k)	O$(deg^+(G) * log\vert P\vert)$
(v, j, k)	O$(log\ deg^+(G) * log\vert P\vert)$

Table 13.3: Triple pattern resolution times on BTWO*.

	Index Order			**Triple Patterns**							
	SP-O	PS-O	OP-S	(i,j,k)	(i,j,v)	(i,v,k)	(i,v,v)	(v,v,v)	(v,j,k)	(v,j,v)	(v,v,k)
BT*	√	-	-	`SP-O`	`SP-O`	`SP-O`	`SP-O`	`SP-O`	-	-	-
BTW*	√	√	-	`SP-O`	`SP-O`	`SP-O`	`SP-O`	`SP-O`	`PS-O`	`PS-O`	-
BTWO*	√	√	√	`SP-O`	`SP-O`	`SP-O`	`SP-O`	`SP-O`	`OP-S`	`PS-O`	`OP-S`

Table 13.4: Summary of indexes and Triple Pattern resolution through incremental proposals.

the given predicate j (Line 4). In such case, it actually retrieves the related subject (Line 5) and outputs it as result.

For instance, the resolution of the pattern $(v, 2, 1)$ in the example in Figure 13.4 starts similar to the previous case, and iterates over the S_o positions 5, 6 and 2 in which the object 1 occurs. For each one, it access the predicate to retrieve the related predicate. If this predicate is equal to the given predicate 2, it retrieves and outputs the subject as results. In the first two positions, it fails retrieving $access(W_p^*, 5) = 1$ and $access(W_p^*, 6) = 1$, as the predicate in both cases is 1. In contrast, the latter 2 position is valid as $access(W_p^*, 2) = 2$, the second asked predicate. Thus, it retrieves the related subject by means of $rank_1(B_o, 1) + 1 = 2$, hence returning $subject = 2$.

Finally, note that, although the predicate test is made sequentially on the list of positions of the object k, one could reduce the number of checks: Once the elements in S_{oP} are ordered by predicate ID, a binary search can be made in the list for object k. The condition of this search, is that the retrieved predicate of each position is less, equal, or higher to the given predicate j. This way we can reduce the number of comparison logarithmically with respect to the number of the size of lists in S_{oP}.

Algorithmic costs. The `OP-S` costs are perfectly described by the aforementioned in-degree metrics, as they characterize the cardinality of objects. In particular, the cost of the $occsObj(j)$ primitive can be parametrizes as follows:

- $occsObj(k)$: It performs two `select` operations over the bitsequence B_{oP}, $O(1)$, and for each occurrence it retrieves its position with a `rank` operation on the bitsequence B_o, $O(1)$. The number of occurrences of an object is perfectly parametrized by its "in-degree" $deg^+(k)$, hence this primitive runs in time O($deg^+(k)$).The general case runs in an average time O($\overline{deg^+}(G)$).

As can be seen, the cost of retrieving all occurrences of an object is proportional to the number of occurrences. This was obviously expected as the adjacency list in *O-Index* groups all these occurrences. Then, the cost of the TP resolution presented above can be summarized as follows:

- (v, v, k) - Algorithm 12: This algorithm first calls the $ocssObj(k)$ primitive (Line 1), O($deg^+(k)$). Next, for each retrieved position, it accesses the Wavelet Tree W$_p$, O($log|P|$) and uses a `rank` operation over the bitmaps, O(1). As for previous cases, we can assume an efficient implementation which obviates the loop over the positions (Line 2) as part of $occsObj$ code (line 4). Thus, this resolution runs in time O($deg^{+}(k) * log|P|$). For the general case, this TP runs in an average time O($deg^{+}(G) * log|P|$).

- (v, j, k) - Algorithm 13: The algorithm performs similar to the previous case. It first calls the $ocssObj(k)$ primitive (Line 1), O($deg^{+}(k)$). Next, for each retrieved position, it accesses the Wavelet Tree W$_p$, O($log|P|$). In contrast to the previous resolution, it only retrieve subjects, in O(1) when the predicate is exactly j. Nevertheless, the time remains in O($deg^{+}(k) * log|P|$). For the general case, this TP runs in an average time O($deg^{+}(G) * log|P|$).

  Note that, the aforementioned optimization performing a binary search in the object occurrences can significantly reduce this time. With this optimization, we do not iterate on all the $deg^{+}(k)$ occurrences of the object k, but we perform a binary search in logarithmic time. Thus, the time decreases to O($log\ deg^{+}(k) * log|P|$). For the general case, this optimization runs in an average time O($log\ deg^{+}(G) * log|P|$).

Table 13.3 represents the resolution times of all TP in BTWO*. Note that, compared to the times for BTW* (Table 13.2) we have updated the cost of resolving (v, j, k), as As we have shown, the *O-Index* provides a more efficient time. In particular, one can see that this time was previously O($log|P| * (\overline{deg_P^{+}}(G) + 1) + log(\overline{deg^{--}}(G))$) versus the novel O($log\ deg^{+}(G) * log|P|$) with the *O-Index*.

One could effectively assume that $log\ deg^{+}(G) << (\overline{deg_P^{+}}(G) + 1) + log(\overline{deg^{--}}(G))$. In plain words, the logarithm of the mean number of triples in which an object appears is much less than the average number of subjects in which a predicate occurs.

Application. The *O-Index* enhancement contributes with an index `OP-S` and allows access and TP by object to be efficiently performed (latest two rows in Table 13.3). Two remarks should be done. First, we have stated that this efficient performance is at the cost of a non-negligible space overhead. In particular, the *O-Index* adds an overhead of $n\lceil logn\rceil + o(n)$ bits, although we will justify in the empirical study (Section 13.3) that the total size remains small in comparison to other competitive solutions. We also remark that this *O-Index* accesses the Wavelet Tree and this adds an extra $log|P|$ term in the resolution. This extra term is present in the latest two rows in Table 13.3. Thus, if access by predicate is not required, one configuration could maintain the stream of predicates S_p loaded in an array (with access O(1)).

Table 13.4 summarizes the included indexes as well as the main index used to resolve each TP variant. In short, BTWO* resolves all combination of TP. Note that, in accordance with Arias et al. (2011), it covers a mean of up to 66% of all queries in *Dbpedia* and 97% in the *Semantic Web Dog Food* log. That is, the presented BTWO* configuration, without additional optimizations nor query planners, is able to efficiently resolve more than all 80% of the SPARQL queries[7].

In addition, resolving all TP variants implies that BTWO* holds the basis to resolve joins of triple patterns (the SPARQL BGPs formalized in Definition 4) and thus all SPARQL queries from the point of view of triple indexing.

A compressed alternative for the *O-Index*. We end this section with an additional remark on the *O-Index*. Note that this index does not replace the original stream of objects, but it is constructed on top of it with a significant space overhead. Thus, we point out that other alternative structures could be considered for the required `OP-S` index, exploiting the space/performance tradeoff. In the following, we provide brief notes on BTWO-GMR*, a more compact representation at the cost of performance degradation.

BTWO-GMR* substitutes the previous *O-Index*, using instead a succinct structure performing `rank`, `select` and `access` operations on the objects in $\mathcal{S}_o$. In particular, we propose to load $\mathcal{S}_o$ on a *GMR*

[7]This is the average in *Dbpedia* and the *Semantic Web Dog* in tune with Arias et al. (2011).

Algorithm 14 `occsObj(k)` in BTWO-GMR*

```
1: function OCCSOBJ(k)
2:     $numOccs \leftarrow \mathbf{rank_k}(\mathcal{G}_o, \mathcal{G}_o.size())$;
3:     for ($x = 1$ to $numOccs$); do
4:         $posObj[\,] \leftarrow \mathbf{select_k}(\mathcal{G}_o, x)$;
5:     end for
        return $posObj$;
6: end function
```

structure (Golynski et al., 2007) (see the definition in Section 2.4.2). As we stated, this structure performs efficiently on large alphabets (in contrast to other alternatives such as the Wavelet Trees).

Definition 33 (BTWO-GMR*) *The Bitmap Triples configuration at consumption time enhanced with a Wavelet Tree index and a* GMR *structure, denoted* BTWO-GMR**, is the succinct bitsequence indexes $\mathcal{B}_p^*$ and $\mathcal{B}_o^*$, the succinct Wavelet Tree $\mathcal{W}_p$ and the* GMR *structure $\mathcal{G}_o$.*

Similar to the Wavelet Tree, the *GMR* structure $\mathcal{G}_o$ serves the following operations :

- `rank`$_k(\mathcal{G}_o, m)$ counts the occurrences of the object k in $\mathcal{G}_o[1, m]$.
- `select`$_k(\mathcal{G}_o, m)$ locates the position for the m-th occurrence of the object k in $\mathcal{G}_o$.
- `access`$(\mathcal{G}_o, m)$ returns the symbol in $\mathcal{G}_o[m]$.

The $\mathcal{G}_o$ structure uses $nlog\sigma + o(nlog\sigma)$ bits, but it fully replaces the original $\mathcal{S}_o$ stream. Regarding its performance, we consider the *GMR* representation which supports `access` and `rank` in $O(loglog\sigma)$, being $\sigma = |O|$, and `select` in $O(1)$ (see our basic concepts in Section 2.4.2 and the original proposal by Golynski et al. (2007) for additional details). This decision is based on the resolution shown in the following, which makes extensive use of `select` operations.

It is worth noting that the resolution of triple patterns by object in BTWO-GMR* is performed very similar than in BTWO*. In fact, Algorithms 12 and 13 run exactly similar. The only difference is that we replace the `occsObj(k)` function by the corresponding object retrieval in $\mathcal{G}_o$. This substitution is illustrated in Algorithm 14, and the operative is very similar to the previous `occsPred` function in the Wavelet Tree (see Algorithm 8). As can be seen, a `rank` operation over the sequence returns the number of occurrences (Line 2) and, for each one (Line 3), we retrieve the position of the occurrence in $\mathcal{G}_o$ with a `select` operation.

Without going into more details, one can easily see that the general performance degradation in BTWO-GMR*, compared with BTWO*, is due to two main reasons:

- In BTWO* we directly retrieve an object in O(1) by accessing the array of objects in $\mathcal{S}_o$. In contrast, BTWO-GMR* has to perform an `access` operation over $\mathcal{G}_o$ in $O(loglog\sigma)$, being $\sigma = |O|$.
- The operation `occsObj` is also slightly faster in BTWO*. It runs in an average time O($\overline{deg^+}(G)$), that is, proportional to the mean number of occurrences of an object. In contrast, BTWO-GMR* performs one `rank` operation and then one `select` operation per occurrence. Thus, the general case in BTWO-GMR* runs in an average time O($loglog\sigma + \overline{deg_P^+}(G)$)).

As we will show in the experiments (§13.3.5), the BTWO-GMR* performance degradation is moderate and it can be perfectly assumed by many solutions. In particular, BTWO-GMR* provides a good space/tradeoff opportunity for those applications which show more restricted spatial requirements.

Dataset	Original Size	Triples		
	(MB)	PT	CT	BT
SWDF	16	2.93%	2.65%	**1.76%**
2011 Australian Census	52	2.82%	2.63%	**1.99%**
Jamendo	144	3.73%	3.51%	**2.18%**
AEMET	726	2.56%	2.49%	**1.54%**
LinkedMDB	850	4.22%	4.12%	**2.60%**
Wordnet	974	3.75%	3.57%	**2.23%**
Affymetrix	6,526	4.20%	3.67%	**2.49%**
Flickr	6,714	4.53%	3.80%	**2.55%**
Dbtune	9,566	4.19%	4.02%	**2.49%**
DBLP	9,799	3.80%	3.32%	**2.15%**
2000 US Census	21,796	4.81%	4.87%	**3.00%**
Linked Geo Data	39,423	5.58%	5.47%	**3.46%**
Dbpedia 3-8	63,053	5.55%	3.77%	**2.92%**
Ike	102,662	3.47%	3.31%	**1.95%**

Table 13.5: Compression ratio of Bitmap Triples (BT) component *w.r.t* the original size of each dataset, in comparison with Plain and Compact Triples.

13.3 Experimental Evaluation

In this section, we evaluate the size and TP query performance of the proposed indexes on top of HDT. We make use of the corpora we are employing in the rest of the thesis, described in Section 4.2.

First, we briefly study the size of the Bitmap Triples representation in comparison with Plain and Compact Triples (§13.3.1). We also measure the space overhead of the *BT**, *BTW**, and *BTWO** succinct indexes (§13.3.2). Next, we compare the *BTWO** performance at consumption with two indexes from the state of the art (§13.3.3). These tests are performed on the "consumer" computer presented in Section 7.4, reporting *user* times. Finally, we analyze the impact of alternative orderings for the triples (§13.3.4) and the BTWO-GMR* variation (§13.3.5), in size and query performance.

All sources are developed in C++ and compiled on g++ 4.7.2 with `-O9` optimization. We use the bitmap and Wavelet Tree structure from `libcds` (*Compact Data Structures Library (libcds)*, 2012). The parametrization will be referred in each experiment.

13.3.1 Bitmap Triples Compression

We first analyze the impact of our Bitmap Triples (BT) configuration in the HDT representation. Table 13.5 shows the compression ratio of BT over the total size of the dataset (in N-Triples). We compare BT with respect to the Plain Triples (PT) and Compact Triples (CT) representations presented in Section 7.2.3. As can be seen, BT achieves the most compressed representation for the underlying graph, clearly outperforming Compact Triples: BT size is about 60% the size of CT and up to 50% the size of PT.

We take up again the evaluation performed in Section 7.4.1, establishing a comparison when using Bitmap Triples in the HDT representation. Thus, Table 13.6 compares HDT with universal compressors (gzip and bzip2). Note that we do not use an advanced functional dictionary (such as $\mathcal{D}_{comp}$), but a plain dictionary encoding of references (see Section 7.2.2). We also codify the ID-triples with *log* bits (of the corresponding number of elements). As expected, Table 13.6 shows that HDT with Bitmap Triples achieves the most compressed ratios, being 10% smaller (on average) than HDT with the Compact Triples variant. Again, compression ratios are only around 2 times bigger (on average) than those for gzip, demonstrating the ability of HDT to obtain compact representations of RDF .

Dataset	Triples (millions)	Size (MB)	HDT			Universal Compressors	
			PT	CT	BT	gzip	bzip2
SWDF	0.1	16	18.21%	17.92%	17.02%	9.68%	6.63%
2011 Australian Census	0.4	52	8.70%	8.51%	7.87%	2.80%	1.33%
Jamendo	1.0	144	24.87%	24.64%	23.31%	5.83%	4.16%
AEMET	3.5	726	13.77%	13.69%	12.74%	2.57%	1.20%
LinkedMDB	6.1	850	15.89%	15.79%	14.26%	4.75%	2.79%
Wordnet	6.3	974	12.85%	12.66%	11.32%	4.97%	3.22%
Affymetrix	44.2	6,526	16.17%	15.64%	14.46%	5.42%	3.43%
Flickr	49.1	6,714	16.58%	15.84%	14.60%	9.03%	7.40%
Dbtune	58.9	9,566	14.57%	14.41%	12.87%	11.24%	7.65%
DBLP	60.1	9,799	20.62%	20.14%	18.97%	5.42%	3.49%
2000 US Census	149.2	21,796	7.45%	7.50%	5.63%	4.62%	2.27%
Linked Geo Data	274.7	39,423	27.07%	26.96%	24.95%	5.90%	4.13%
Dbpedia 3-8	431.4	63,053	18.32%	16.55%	15.70%	8.01%	5.90%
Ike	514.8	102,662	11.86%	11.71%	10.34%	3.22%	1.08%

Table 13.6: Compression ratio of HDT with Plain, Compact and Bitmap Triples, and universal compressors results.

It is worth mentioning that, to boost exchanging, two methods can achieve the most compressed representation for HDT (outperforming traditional text compression for RDF). On the one hand, we proposed an "additional HDT Compression" in Section 7.4.3, which applies text compression of the HDT representation. This already outperformed text compression, and can also be applied when using Bitmap Triples. On the other hand, compressed RDF dictionaries, such as $\mathcal{D}_{comp}$, can encode the Dictionary component. We experiment with this possibility in the next part of this thesis.

13.3.2 Analyzing the Space Overhead of *BTWO**

Table 13.7 reflects the space requirements of the particular indexes in BTWO* when loaded at consumption time. We provide the size of the indexes with respect to the Bitmap Triples size in each dataset. Thus, the second column corresponds to the size of the bitmaps ($\mathcal{B}_p^*$ and $\mathcal{B}_o^*$) introduced since the BT* configuration. The third column considers the size of the Wavelet Tree ($\mathcal{W}_p^*$) introduced since the BTW* configuration. Finally, the OP-S index of BTWO* is presented in the fourth column. Note that our implementation uses RG bitmaps (sampling of 20) both for our bitmap indexes and the Wavelet Tree.

Several comments can be drawn from these results. First, the BT* bitmaps required to act as an SP-O index (as summarized in Table 13.4) are only 8% (on average) the size of the Bitmap Triples. That is, the consumer can resolve about 50% of the most common queries in SPARQL (see Section 13.1.3), with a little 8% overhead over the transferred triples representation.

In turn, the third column in Table 13.7 shows that a mean of 20% of space overhead is required to build the Wavelet Tree (PS-O index). As stated, in the final BTWO* configuration, this index contributes to resolve the (v,j,k) patterns which, in practice, are not massively used (see Arias et al. (2011)). In addition, it adds a logarithmic cost to access the predicates. We will discuss in the next Chapter that this index could be obviated if such type of access is not required.

Finally, the *O-Index* size is provided in the fourth column. It is easy to see, though, that this index supposes a significant space overhead, around 84% of the original BT size. The *O-Index* size covers the array of positions ($\mathcal{S}_{oP}$) of length n (the number of triples) and its bitmap index ($\mathcal{B}_{oP}$). Nonetheless, the *O-Index* i) completes the index structure at consumption time, ii) it resolves the common (v,j,v) and (v,v,k) patterns (Arias et al., 2011), and iii) it performs efficiently (see the evaluation in Section 13.3.3).

Dataset	Bitmap Triples	Indexes		
	(MB)	BT* bitmaps	BTW* Wavelet Tree	*BTWO* O-Index*
SWDF	0.28	10.41%	27.08%	80.44%
2011 Australian Census	1.04	11.45%	21.87%	84.92%
Jamendo	3.14	9.99%	17.27%	85.05%
AEMET	11.18	9.82%	17.65%	88.39%
LinkedMDB	22.10	8.64%	24.91%	80.84%
Wordnet	21.70	8.84%	21.99%	83.79%
Affymetrix	162.24	7.52%	16.33%	85.67%
Flickr	171.37	7.77%	11.74%	90.10%
Dbtune	237.98	7.61%	24.43%	80.80%
DBLP	210.46	7.83%	12.00%	93.25%
2000 US Census	654.24	7.39%	25.08%	79.85%
Linked Geo Data	1,362.76	6.12%	32.77%	69.68%
Dbpedia 3-8	1,841.18	5.47%	19.95%	82.06%
Ike	1,997.88	7.98%	11.46%	93.31%
MEAN	-	8.35%	20.32%	84.15%

Table 13.7: Space requirements of the indexes BT* bitmap indexes, the BTW* Wavelet Tree and the BTWO* *O-Index*, given as ratios (in %) *w.r.t* the original Bitmap Triples size of each dataset.

Table 13.8 shows the total sizes of the incremental configurations, which are a direct consequence of the index sizes presented above. Thus, BT* adds the bitmap overhead directly to the BT size, resulting in a mean of 8% space overhead. In turn, the Wavelet Tree may contribute with an important overhead in BTW* but, as can be seen, BTW* only adds a 9.33% overhead over the BT size (on average). The reason is simple: the Wavelet Tree do not append its overhead but it completely replaces[8] the integer sequence S_p by the Wavelet Tree W_p. This assures that, thanks to this succinct structure, we can provide a `PS-O` index with a very limited overhead. Finally, the `OP-S` index overhead is also directly added to the BT size, resulting in a total of around 93% space overhead (on average) with respect to the BT size.

In summary, the final BTWO* configuration adds three indexes (SP-O, PS-O and OP-S) and provides total TP resolution with a mean of 93% space overhead of the exchanged Bitmap Triples component.

13.3.3 BTWO* Performance Comparison

In the following, we analyze the performance of BTWO* with respect to the state of the art. In particular, we compare and analyze the representation space and the triple pattern resolution performance against RDF3X (Neumann & Weikum, 2010) and k^2-triples (Álvarez-García et al., 2011). Section 12.2.2 includes a detailed review of both solutions. RDF3X is a native multi-indexing solution on the basis of B^+-trees. k^2-triples follows a vertical partitioning strategy, creating a K^2-tree index per predicate. We also test the improved k^2-triples+ (Álvarez-García et al., 2013) which includes additional SP and OP indexes. Thus, it addresses better performance at the cost of additional space overheads.

We first compare the space requirements of each solution. We choose again a subset of six datasets from our evaluation setup in Section 4.2. These datasets correspond to that used for evaluating $\mathcal{D}_{comp}$ as they cover different application domains and number of triples. Table 13.9 shows the space ratio with respect to the original Plain Triples size, that is, three IDs per triple (in log bits). As can be seen, the k^2-triples solution takes advantage of the sparse distributions per predicate, leading to the most compressed solution for RDF triple indexing. On average, k^2-triples outperforms 7 times our BTWO* compression and up to 42 times the results of RDF3X. The improved k^2-triples+ proposal, including SP and OP

[8]Note that S_p is destroyed after the creation of the Wavelet Tree.

Dataset	Bitmap Triples	Triples		
	`(MB)`	`HDT` BT*	`HDT` BTW*	`HDT` BTWO*
SWDF	0.28	110.41%	111.91%	192.35%
2011 Australian Census	1.04	111.45%	112.51%	197.42%
Jamendo	3.14	109.99%	110.82%	195.87%
AEMET	11.18	109.82%	110.66%	199.05%
LinkedMDB	22.10	108.64%	109.83%	190.67%
Wordnet	21.70	108.84%	109.89%	193.67%
Affymetrix	162.24	107.52%	108.30%	193.97%
Flickr	171.37	107.77%	108.33%	198.43%
Dbtune	237.98	107.61%	108.78%	189.57%
DBLP	210.46	107.83%	108.40%	201.65%
2000 US Census	654.24	107.39%	108.58%	188.43%
Linked Geo Data	1,362.76	106.12%	107.69%	177.37%
Dbpedia 3-8	1,841.18	105.47%	106.43%	188.49%
Ike	1,997.88	107.98%	108.52%	201.83%
MEAN	-	108.35%	109.33%	193.48%

Table 13.8: Total space requirements of BT*, BTW* and BTWO* *w.r.t* the original Bitmap Triples size.

Dataset	Plain Triples	In-memory configuration			
	Size `(MB)`	`HDT` BTWO*	k^2-Triples	k^2-Triples+	RDF3X
2011 Australian Census	1.47	139.33%	13.83%	15.84%	681.81%
Jamendo	5.38	114.48%	13.52%	23.61%	993.74%
AEMET	18.61	119.64%	8.07%	11.13%	665.20%
Dbtune	400.36	112.68%	38.06%	46.85%	673.46%
2000 US Census	1,049.25	117.49%	33.09%	39.50%	508.73%
Dbpedia 3-8	3,497.36	99.23%	38.56%	51.00%	570.97%

Table 13.9: Space requirements *w.r.t* the original Plain Triples size (in log. bits) of each dataset.

indexes, is 5 times more compressed than BTWO*. Nonetheless, we will show below that BTWO* performs several orders of magnitude better than both k^2-triples proposals. In turn, the BTWO* solution uses almost 6 times less space than RDF3X, on average. Note also that BTWO* is only slightly bigger than the Plain Triples size, except for *Dbpedia* which is even smaller (99.23%).

Then, we analyze the retrieval ability of our BTWO* solution. To do so, we evaluate the performance on triple pattern solution as it is the core for BGP resolution in SPARQL. The query testbed consists of randomly generated TP: for each dataset, we consider 1,000 random triple patterns of each type. Nonetheless, note that the type (v,p,v) is limited by the number of different predicates[9].

Figures 13.5 and 13.6 show the resolution times for *Dbtune* and *Dbpedia* respectively. We design a warm scenario for the RDF3X on-disk solution in order to reduce the penalization of the I/O transactions *w.r.t.* the in-memory solutions k^2-triples and BTWO*. Thus, RDF3X figures report the mean resolution time of six consecutive repetitions of each query, forcing results to be available in main memory.

Several remarks can be drawn from Figures 13.5 and 13.6. The most important remark is that, in general terms, BTWO* clearly outperforms k^2-triples and RDF3X on both datasets. In particular, BTWO* excels in most triple patterns, improving RDF3X by 1 level of magnitude and up to 3 for k^2-triples. Let us particularize the analysis by BTWO* indexes and the triple pattern resolution.

[9]For *Dbpedia*, we test all the 57,986 predicates as the random generation could result too limited.

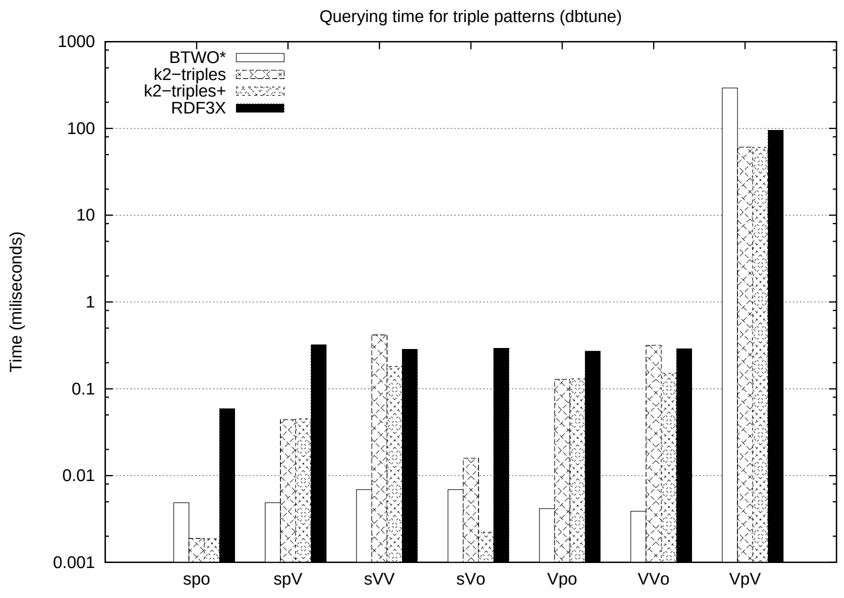


Figure 13.5: `HDT BTWO`* TP query performance in *Dbtune*.

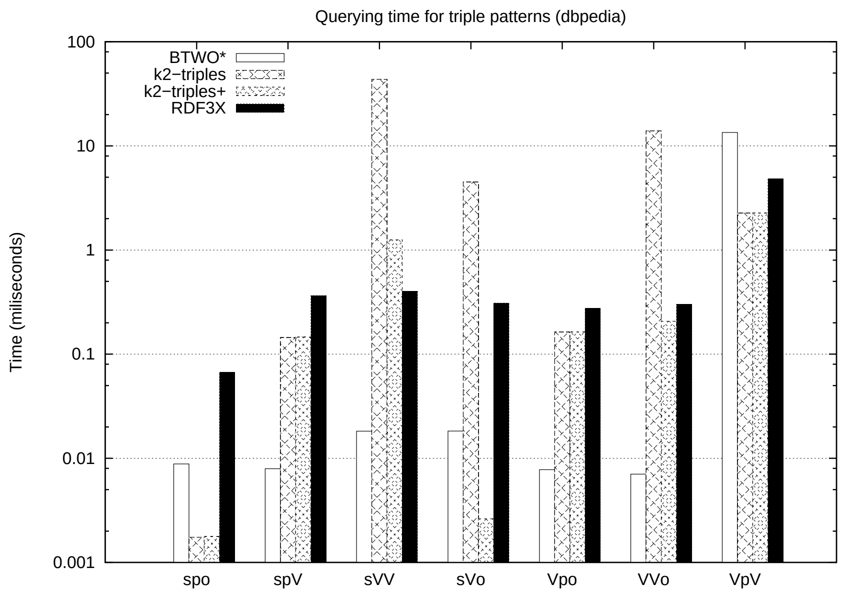


Figure 13.6: `HDT BTWO`* TP query performance in *Dbpedia*.

- BT* index - *access by subject*. This corresponds to the triple patterns (s,p,o), (s,p,V), (s,V,V), and (s,V,o) in the figures. As stated, BT* SP-O ordering favors the access by subject, hence BTWO* excels in these triple patterns. For instance, in *Dbtune*, BTWO* resolves (s,p,V) 9 and 66 times faster than k^2-triples and RDF3X respectively. As expected, k^2-triples pays its vertical partitioning overload with unbounded predicates, as all matrix have to be queried in these cases: in *Dbpedia*, k^2-triples performs 2397 and 246 times slower than BTWO* in (s,V,V) and (s,V,o) respectively. However, the additional indexes in k^2-triples+ significantly reduce this difference. In fact, k^2-triples+ resolves (s,V,o) 7 times faster than BTWO*. In addition, one can see that k^2-triples also outperforms BTWO* in (s,p,o) resolution, *i.e.*, those queries checking the existence of a triple. In this case, BTWO* performs two binary searches (see Section 13.1.2) whereas k^2-triples uses its optimized operation of checking a cell in the S-O adjacency matrix of the given predicate.

- `O`-index - *access by object*. This index accesses the triples by object and, thus, it helps resolve the triple patterns (V,p,o) and (V,V,o) in the figures. As can be seen, BTWO* clearly emerges as the fastest solution resolving these triple patterns, beating the other proposals by several orders of magnitude. For instance, in *Dbpedia*, BTWO* is 1979 and 43 times faster than k^2-triples and RDF3X respectively for the (V,V,o) triple pattern. Although the additional indexes in k^2-triples+ improve its performance, BTWO* is still one order of magnitude faster. Note that, as explained in Section 13.2.2, the *O-Index* finds the adjacency list of the given object in constant time (proportional to its number of occurrences).

- Wavelet Tree index - *access by predicate*. The Wavelet Tree W_p in predicates provides a `PS-O` index in BTWO*, resolving the (V,p,V) triple pattern in the figures. As expected, the higher costs of the Wavelet Tree index (logarithmic with the number of predicates as shown in Section 13.2.1) have a noticeable effect in the reported times for (V,p,V): BTWO* is 3 to 6 times slower than the other solutions in both datasets. In this case, k^2-triples reports the best performance once it has to "dump" the adjacency matrix of the given predicate.

Thus, in general, BTWO* reports the best overall performance for RDF retrieval. Taking the mean of the performance (in times faster) per triple pattern and dataset[10], BTWO* runs 33.25 times faster than RDF3X, 344.80 than k^2-triples and 15.32 than k^2-triples+.

13.3.4 BTWO* Order Comparison

We had assumed that BT always keeps the original ordering by Subject-Predicate-Object (SPO) up until now. In fact, this is the logical order according to the notion of RDF triple (or statement). We then study other alternative orders for the BT representation and the subsequent indexes: BT*, BTW* and BTWO*. We first analyze the space requirements of each alternative. Then, we choose the most efficient alternatives to compare the query performance against the traditional SPO ordering.

Space requirements. The space requirements of all the alternative orders are shown in Figure 13.7, in which each bar draws the ratio against the size of the corresponding structure in SPO order. The given ratio is the average of the presented fourteen datasets (see experimental framework in Section 4.2). We summarize below the most important implications of the alternative orders:

- *Subject-Object-Predicate (SOP)*. In this case, we swap the order of the adjacency lists in BT. That is, in the top level of BT we list all the objects related to each subject, and the bottom level represents all the predicates for a given *(subject,object)* pair. Regarding the indexes at consumption, the original Wavelet Tree of predicates is substituted by a Wavelet Tree of objects. In such case, the alphabet of symbols is the number of different objects, which is much bigger than the number of predicates. This overhead is reflected in the size of BTW* and BTWO* in Figure 13.7. For instance, the BTWO* index in SOP order is 1.42 times bigger than those in SPO order.

- *Predicate-Subject-Object(PSO) and Predicate-Object-Subject(POS)*. In both cases, BT acts similar to a Vertical Partitioning technique. For each predicate, its related elements are listed. Note that, as in the previous case, the middle index for POS will be an overloading Wavelet Tree of objects. In contrast, a Wavelet Tree of subjects will be constructed for PSO, which also becomes much bigger than the original Wavelet Tree of predicates. This results in more space requirements, as shown in Figure 13.7: Although the BT structure and the BT* index are comparable in size to those in SPO, the Wavelet Tree overhead in PSO and POS is predominant. Thus, BTW* and BTWO* demand more space than the originals in SPO.

[10]This is equivalent to choosing a TP at random in *Dbtune* or *Dbpedia*.

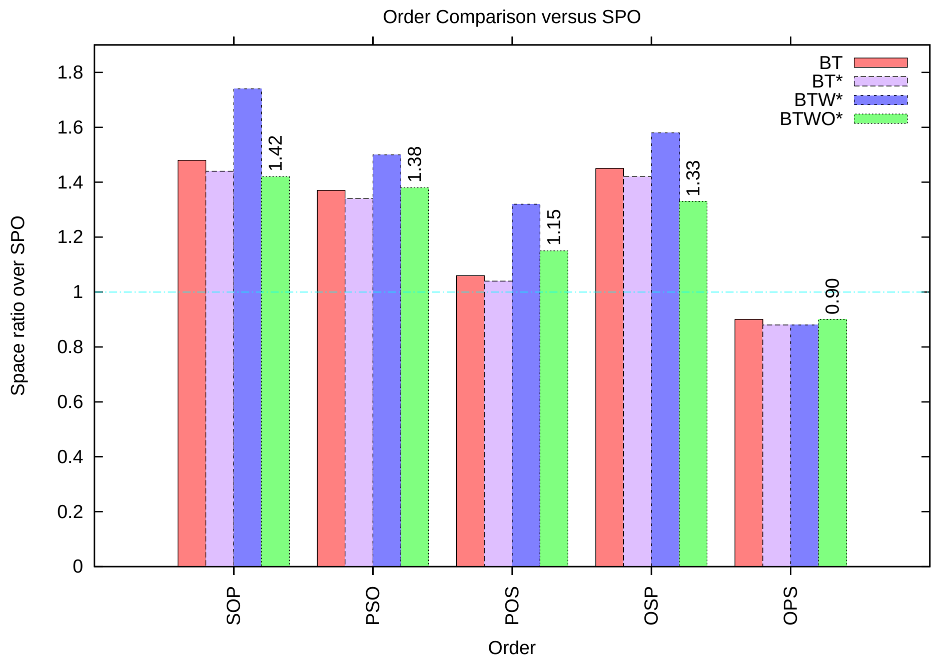
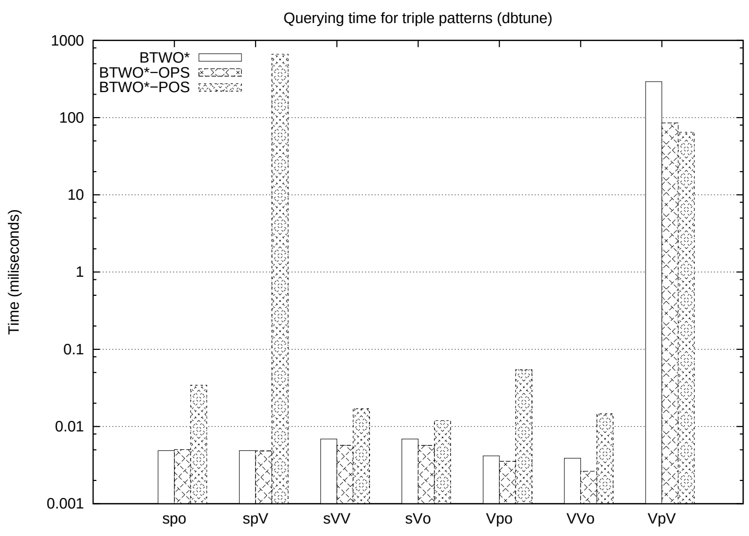


Figure 13.7: Comparison of alternative orders for BT and its indexes at consumption. Each graph represents the ratio against the size of the corresponding structure in SPO order.

Note that all the structures are smaller in POS than in PSO. One should find the cause of this difference in the BT adjacency lists, characterized by our metrics. That is, the size of the lists of objects in POS are delimited by the predicate out-degree: the number of different objects related to given predicates. The bigger is the predicate out-degree, the larger are the lists of objects in POS. In turn, larger lists group more triples and, thus, they yield to more compact representations. Regarding the lists of subjects in PSO, they are delimited by the predicate in-degree, and similar reasoning can be made. In general terms, the mean predicate out-degree is smaller than the corresponding in-degree (see Section 4.3.5), *i.e.*, a predicate is related to more objects than subjects. Thus, the grouping lists in POS are larger than in PSO, resulting in more compact BT and BT*.

- *Object-Subject-Predicate(OSP).* This ordering places the objects at the top of the BT representation, and keeps the lists of subjects related to each object, relegating the predicates at the bottom. As we stated in our experiments, it is common that only one predicate is related to a *(subject,predicate)* pair (see Section 4.3.4). Thus, the BT compact structure is poor (it does not group references) and its size is significant bigger than the SPO ordering, as can be seen in Figure 13.7. In addition, the middle index should be again a Wavelet Tree of subjects which yields to a significant overhead: the BTWO* index is 1.33 times bigger than the corresponding in SPO order.

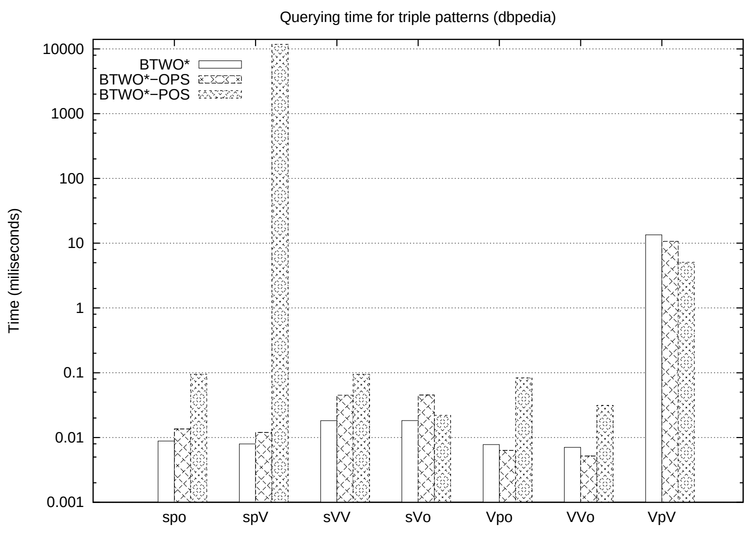
- *Object-Predicate-Subject(OPS).* The OPS ordering is, surprisingly, the most compact alternative. As shown in Figure 13.7, it improves the size of all structures in SPO order, being around 10% more compact. Note that in OPS, as in SPO, the predicate is in the middle position and its adjacency list is indexed with a Wavelet Tree on a short alphabet. The improvement over SPO can be explained simply by the codification of the elements. In BT, the bottom stream always stores n elements, being n the number of triples. These are encoded with a given number of bits: the logarithm of the number of different elements. In SPO, BT codifies n objects with the logarithm of different objects, whereas in OPS it codifies n subjects with the logarithm of different subjects. In general, the number of different subjects is significant smaller than the number of different objects (see Section 4.2). This is the main reason OPS ordering demands smaller space than SPO.



Size ratio (in %) against SPO:

Order	BT	BTWO*
OPS	83.60%	86.48%
POS	95.04%	104.88%

Figure 13.8: `HDT BTWO`* TP query performance in *Dbtune*, comparison between different orders.



Size ratio (in %) against SPO:

Order	BT	BTWO*
OPS	96.80%	98.07%
POS	107.65%	116.08%

Figure 13.9: `HDT BTWO`* TP query performance in *Dbpedia*, comparison between different orders.

Query performance. We choose the two most compact alternatives from the previous study: POS and OPS. The first one is almost as compact as SPO, and it is the representative of a Vertical Partitioning technique in BT. The latter, OPS, is 10% more compact than SPO, and it has exactly the same philosophy but it reverses the order of the elements. Figures 13.8 and 13.9 show the TP resolution time of these two alternative in comparison with the original SPO ordering, in *Dbtune* and *Dbpedia* respectively. Note that we perform over the same previous TP testbed (see Section 13.3.3). The rightmost tables in the figures show, for each dataset, the ratio of the orders against the size in SPO order.

The POS performance reports similar figures in both datasets. As can be seen, it is the worst solution in all cases except for (V,p,V) resolution. In this particular case, the resolution takes advantage of the Vertical Partitioning by predicate, outperforming the other solutions a mean on 2.6 times. In addition, as shown in the rightmost tables, the BTWO* indexes in POS always demands more space requirements than both SPO and OPS orderings. It is also worth noting that (s,p,V) resolution is extraordinary slow with a POS order: the algorithm would retrieve the list of objects related to the given predicate and, for each object, it checks if the given subject is related to the *(predicate,object)* pair. This operation is extremely low once many objects can be related to each predicate (see Section 4.3.5).

All these facts implies that POS is discouraged in favor of the other solutions. Nevertheless, we outline that the POS order could be chosen in some scenarios demanding excellent times in (V,p,V) resolution in spite of the other weakness: a little overhead in space and certain degradation in the rest of queries, in particular for (s,p,V) resolution.

The analysis of OPS performance is more complex. Note that the resolution time in OPS for *Dbtune* is slightly better than SPO, whereas this is not the case for *Dbpedia*. Let us analyze the following cases of triple patterns:

- (V,p,o) and (V,V,o). In these triple patterns the OPS ordering always report the best performance as it indexed the triples by object. In the SPO ordering, both cases are accessed through the BTWO* *O-Index*, in contrast with the faster BT* index used in the OPS case. Algorithmically speaking, one can easily note that the degrees are multiplier factors when resolving (V,p,o) and (V,V,o) in SPO (See Table 13.3). The costs are additive, though, using BT* in OPS.

- (V,p,V). Its resolution in OPS also improves the SPO ordering in the studied datasets. In this case, one can find the reason in the asymmetric degrees of the predicates. That is, (V,p,V) resolution first retrieves all the occurrences of the given predicate in the first stream. Then, for each occurrence, it locates the associated top element in the structure (subjects in SPO or objects in OPS). Last, the corresponding adjacency list of elements (objects in SPO or subjects in OPS) is retrieved. The algorithmic cost is proportional to the number of occurrences of the predicate in the stream, denoted by its predicate in-degree in SPO (see Section 13.2.1), and then to the predicate out-degree in OPS. In general terms, the mean predicate in-degree is bigger than the predicate out-degree (see Figure 4.12 in our experiments), *i.e.* a predicate is related to more subjects than objects. Thus, (V,p,V) resolution costs are also bigger for SPO ordering than for OPS.

- (s,p,o) and (s,V,o). In these patterns, OPS ordering is slightly worse than SPO in both datasets. Note that, in the traditional SPO order, the resolution performs binary searches which depend on the number of predicates per subject and the number of objects related to a *(subject, predicate)* pair (see costs in Table 13.3). In turn, in OPS, this corresponds to costs which are proportional to the number of predicates per object and the number of subjects related to a *(object, predicate)* pair. In *Dbpedia*, an object can appear related to many subjects. This resulted in such poor results that, in fact, we decide to resolve these patterns in OPS starting from the subjects up to top objects. With this decision, performance results are close in SPO and OPS as we perform identically over the same elements, but they are represented in different indexes. SPO ordering slightly outperforms OPS because the latter pays the overload of searching the predicates related to a subject in the bottom index of references.

- (s,p,V) and (s,V,V). Again, OPS improves SPO resolution times in *Dbtune* but they suffer from performance degradation in *Dbpedia*. The reason is partially different than the previous case. In these triple patterns, SPO uses the BT* index whereas the bottom index of subject positions is used in OPS ordering (an *S-Index* in tune with the original *O-Index* in SPO). The resolution with this latter depends on the number of triples in which the subject takes part (the degree as seen in Table 13.3). In a scenario such as *Dbpedia*, with potentially frequent subjects, the OPS is clearly discouraged over SPO.

In summary, as expected, OPS order should be chosen if access by object is prioritized over access by subject. Nonetheless, the rightmost tables in the Figures 13.8 and 13.9 show that OPS ordering is slightly more compact than SPO. This tradeoff places OPS as an interesting candidate in many scenarios at the price of some performance degradation in (s,p,V) and (s,V,V) whenever objects tend to be massively repeated (such in *Dbpedia*).

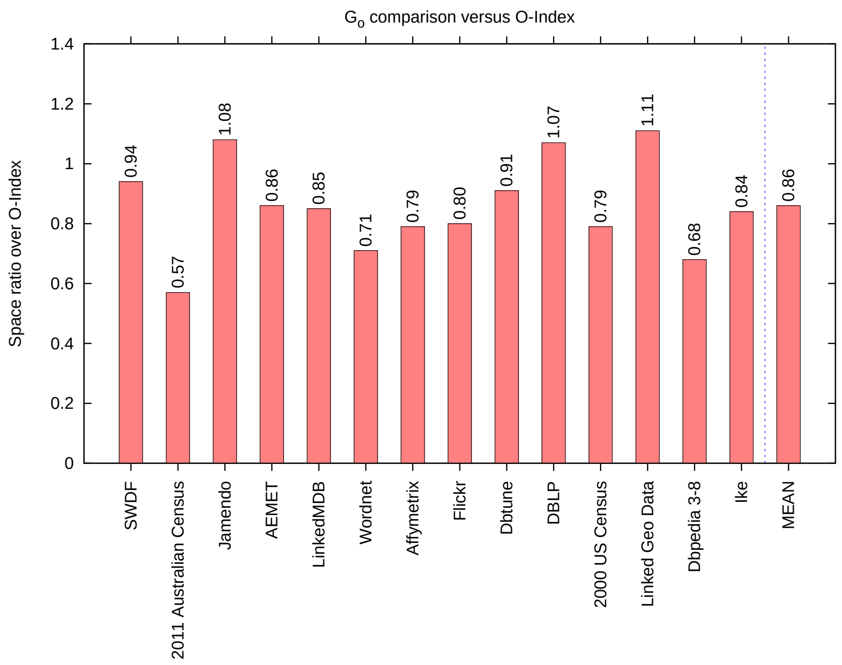


Figure 13.10: Comparison of the $\mathcal{G}_o$ index and the original *O-Index*. Each bar represents the ratio against the size of the *O-Index* structure (the *O-Index* + the object array $\mathcal{S}_o$) (in SPO order).

13.3.5 The BTWO-GMR* Alternative

We end this evaluation with a brief study on the aforementioned *O-Index* alternative, BTWO-GMR* (see Section 13.2.2). We first analyze its space requirements in the normal SPO configuration. Once we have shown the OPS achievements in the previous section, we verify the BTWO-GMR* compressibility on an OPS ordering. We finally test the query performance of BTWO-GMR* on both SPO and OPS orderings.

Space requirements. Figure 13.10 shows the size ratio of the $\mathcal{G}_o$ index against the original *O-Index* structure, in the evaluated datasets (see details in Section 4.2). That is, a value of 0.68 in *Dbpedia* states that the substitute $\mathcal{G}_o$ structure in *Dbpedia* requires 68% the space of the replaced *O-Index* structure. Note that under the size of the replaced *O-Index* structure we include, in fact, the size of all the object structure, that is, $\mathcal{S}_o$, $\mathcal{S}_{oP}$, and $\mathcal{B}_{oP}$. The last column in Figure 13.10 computes the mean of all the datasets.

As can be seen, the $\mathcal{G}_o$ alternative achieves significant space savings: it takes a mean of 86% and up to 57% *w.r.t* the original *O-Index*. In large datasets, these savings can be crucial to scale up applications. For instance, in *Dbpedia*, the $\mathcal{G}_o$ index saves almost 1 GB of consumer main memory.

Nevertheless, it is worth mentioning that, with this alternative, only three particular datasets achieve slightly bigger figures than the *O-Index* (up to 111%): *Jamendo*, *DBLP* and *Linked Geo Data*. We stated that the *O-Index* adds an overhead of $n\lceil logn\rceil + o(n)$ bits (see Section 13.2.2) to the representation. In total, for the original object structure in BTWO*, we also have to consider $n + o(n)$ bits for $\mathcal{B}_o$ and $nlog|O|$ for $\mathcal{S}_o$. In turn, the $\mathcal{G}_o$ index uses $nlog|O| + o(nlog|O|)$ bits. One can easily see that the difference is comparable. Thus, one should find the reason in the *GMR* construction which implicitly hides some parameters (see our basic concepts in Section 2.4.2 and the original proposal by Golynski et al. (2007) for additional details). Without going into too much details, the *GMR* structure used in the $\mathcal{G}_o$ index builds a virtual matrix of $|O|$ rows and n columns. In these three datasets, the vocabulary of different objects is extremely bigger with respect to the total number of triples. For instance, in *Linked Geo Data* there are more than 121 millions of different objects in 274 million triples (see Table 4.2 in Section 4.2) (almost 1 different object each 2 triples). In such special cases, the *GMR* virtual matrix is almost as long as wide, hence the $\mathcal{G}_o$ index achieves comparable ratios than the original *O-Index*.

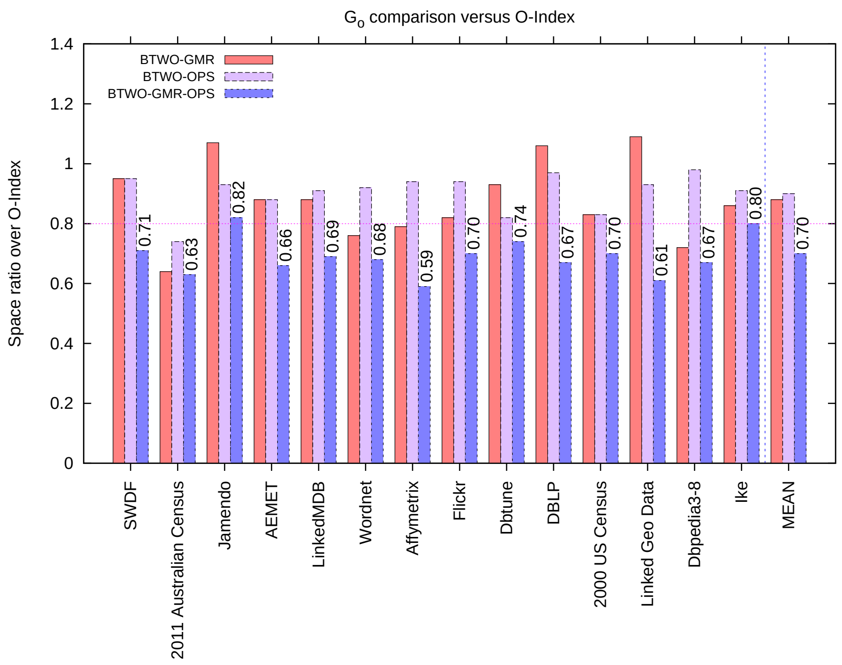


Figure 13.11: Comparison between the BTWO-GMR* and the BTWO* configuration in SPO and OPS orderings. Each graph represents the ratio against the size of BTWO*.

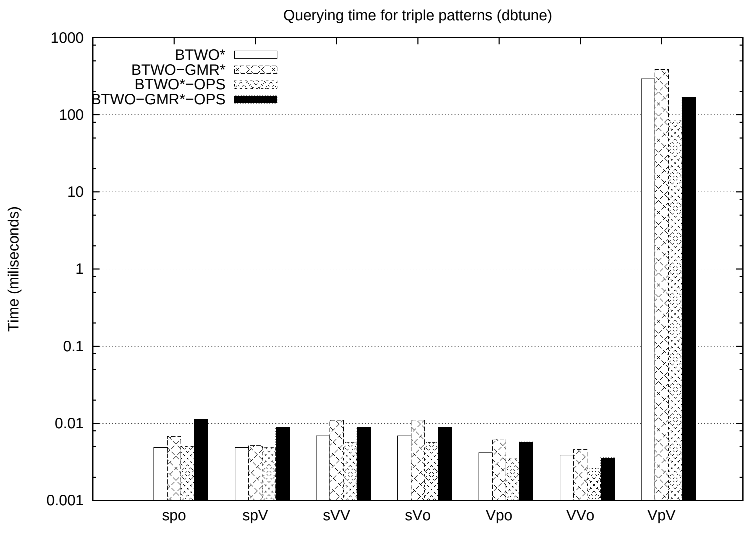
Figure 13.11 first compares the final BTWO-GMR* configuration size, which makes use of the $\mathcal{G}_o$ index instead of the original *O-Index*. This corresponds to the first bar in Figure 13.11 for all datasets, given as a ratio against the original BTWO-GMR* configuration size. The last group of bars represents the mean of all datasets. As expected, the ratios follow the same tendency of the $\mathcal{G}_o$ savings: BTWO-GMR* takes a mean of 88% of the space of the original BTWO*.

Finally, we study the impact of BTWO-GMR* on an OPS ordering, once we established that this ordering also achieves large savings (§13.3.4). Note that, in OPS, the *GMR* index is built on the subjects (at the bottom of the representation). Thus, *GMR* also acts on a large alphabet and a good behavior could also be expected. The second and third bars in Figure 13.11represent the respective space ratios of BTWO* and BTWO-GMR* both in OPS order. These ratios are given against BTWO* on SPO order. As can be seen, BTWO-GMR* on OPS order (referred to as *BTWO-GMR*-OPS*) largely outperforms BTWO* whether on OPS or on SPO order. In fact, it achieves an extraordinary mean compression: in general, BTWO-GMR* on OPS order uses 70% of the original BTWO* size (in SPO order). In *Dbpedia*, for instance, this saves up 1.1 GB of consumer main memory. In turn, in *Linked Geo Data*, which was a corner case for the *GMR* solution in SPO order, BTWO-GMR*-OPS saves more than 744 MB. All this makes BTWO-GMR*, and particularly BTWO-GMR*-OPS, the best candidate for those applications with tight space requirements.

In the following, we test if the *GMR* savings are, as expected, at the cost of performance degradation.

Query performance. We end this section evaluating the *GMR* alternative in TP resolution. To do so, we use the same previous TP testbed (see Section 13.3.3), and we perform on BTWO-GMR* (SPO ordering), as well as over the OPS ordering, BTWO-GMR*-OPS. Figures 13.12 and 13.13 show the TP resolution times. To establish a comparison, the figures also include the aforementioned results for BTWO* and BTWO*-OPS. The rightmost tables in the figures represent the size ratio of each solution against the size of the common BTWO* approach (similar to Figure 13.11).

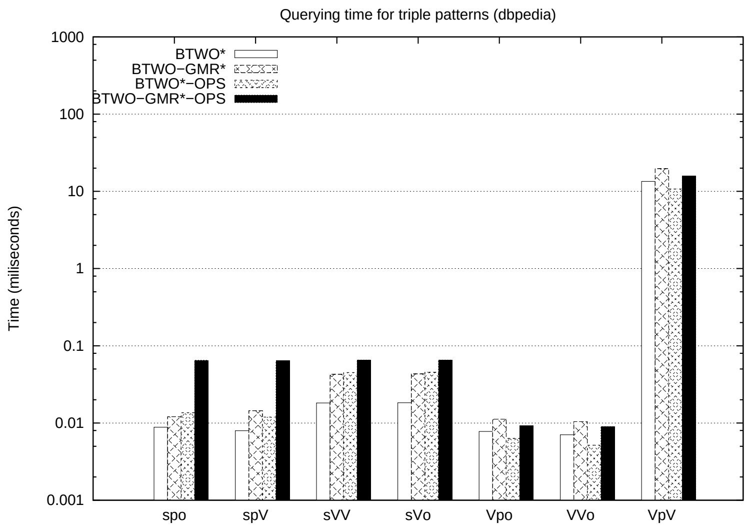
Let us compare first the results of BTWO* and BTWO-GMR*. We analyze two categories: triple patterns by object, thus making use of the particularities of the $\mathcal{G}_o$ index, and the rest of TPs.



Size ratio (in %) against BTWO*:

Structure	Size ratio
BTWO-GMR*	92.67%
BTWO*-OPS	86.48%
BTWO-GMR*-OPS	73.64%

Figure 13.12: `HDT BTWO`* TP query performance of BTWO-GMR* in *Dbtune*.



Size ratio (in %) against BTWO*:

Structure	Size ratio
BTWO-GMR*	72.05%
BTWO*-OPS	98.07%
BTWO-GMR*-OPS	67.48%

Figure 13.13: `HDT BTWO`* TP query performance of BTWO-GMR* in *Dbpedia*.

- (V,p,o) and (V,V,o). In these triple patterns, the BTWO-GMR* alternative uses the $\mathcal{G}_o$ index to retrieve the object occurrences, in contrast to the *O-index* used in the traditional BTWO* configuration. Thus, the difference in the resolution time is solely due to the different performance of both indexes. As can be seen, the theoretical degradation of this alternative (studied in Section 13.2.2) is shown in practice. Nonetheless, the degradation is very moderate in both datasets: the resolution time of these patterns is 40% slower in BTWO-GMR* than in BTWO*, on average.

- (s,p,o), (s,p,V), (s,V,V), (s,V,o) and (V,p,V). As stated, in the rest of the triple patterns, the $\mathcal{G}_o$ index introduces a theoretical slight degradation when accessing the objects (see Section 13.2.2). Obviously, the more objects are accessed in a TP, the more important is the degradation. This can be appreciated in the figures, as the degradation of BTWO-GMR* is noticeable bigger in *Dbpedia* TPs, which access more objects, than in *Dbtune*. On average, the resolution time of these patterns is 63% slower in BTWO-GMR* than in BTWO*.

Surprisingly, the performance degradation due to the $\mathcal{G}_o$ index is slightly more pronounced in those triple patterns which do not access by object. Averaging over all the TP resolutions in both datasets,

Structure	Size Ratio (in %)	Mean Performance Ratio	Performance Ratio by Triple Pattern						
			(s,p,o)	(s,p,V)	(s,V,V)	(s,V,o)	(V,p,o)	(V,V,o)	(V,p,V)
BTWO-GMR*	84%	1.8	1.5	1.3	1.9	1.8	3.3	1.6	1.4
BTWO*-OPS	87%	0.8	1.0	1.0	1.0	1.0	0.9	0.7	0.3
BTWO-GMR*-OPS	70%	2.1	3.4	3.3	2.0	2.0	2.0	1.2	0.6

Table 13.10: Space/performance tradeoffs of BTWO* variants. Average of six datasets (see Table 10.1).

BTWO-GMR* is 56% slower than BTWO*. In any case, the reported times are comparable and this BTWO-GMR* performance degradation could be perfectly assumed by many applications. The size ratios shown in the rightmost tables in Figures 13.12 and 13.13 complete the analysis of the interesting space/performance tradeoff of BTWO-GMR*. For instance, BTWO-GMR* only demands 72.05% of the space required by BTWO* in *Dbpedia*. Thus, an application running on *Dbpedia* can save up to 970 MB of consumer main memory at the cost of 60% slower TP resolution.

Next, we compare the results on OPS ordering represented in Figures 13.12 and 13.13. Note that, in this case, the *GMR*-based $\mathcal{G}_o$ index is built on the subject stream, replacing it.

- (s,p,o), (s,p,V), (s,V,V) and (s,V,o). In OPS ordering, all these cases are resolved accessing the index structure by subject (see Section 13.3.4). That is, the BTWO-GMR*-OPS alternative uses the bottom *GMR*-based index to retrieve all the subject occurrences. As can be seen, in both datasets the reported times of all four TPs are close. In fact, the performance on BTWO-GMR*-OPS for (s,p,o) and (s,p,V) is noticeable worse than on BTWO*-OPS. Note that, on average, a subject has more occurrences than an object (see Table 4.2 in Chapter 4). As we perform proportional to the number of occurrences, thus the degradation of the *GMR*-based index is more pronounced in OPS (acting on subjects) than in SPO ordering (acting on objects). Taking the mean of the performance in both datasets, BTWO-GMR*-OPS is 2.5 times slower than the BTWO*-OPS configuration, but 3.7 times slower than BTWO-GMR* (in SPO ordering).

- (V,p,o), (V,V,o) and (V,p,V). In the first two cases, an OPS ordering makes use of the BT* index to first retrieve the object occurrences. In the latter, the predicate Wavelet Tree is used. Thus, in any case the theoretical degradation is due to the subject access of the *GMR*-based index. This is actually reported in the figures: On average, the BTWO-GMR*-OPS alternative is 60% slower than the BTWO*-OPS configuration, and only 8.5% slower than BTWO-GMR* (in SPO ordering).

As can be seen, the *GMR*-based index on OPS ordering suffers from performance degradation specially for those TP accessing by subject. Averaging over all TPs, the BTWO-GMR*-OPS configuration is 2.5 times slower than BTWO*-OPS, and 2.1 times slower than the BTWO* (in SPO order). Nevertheless, the rightmost tables in Figures 13.12 and 13.13 show the high compression ratios of BTWO-GMR*-OPS: it takes 73.64% and 67.48% the size for BTWO* in *Dbtune* and *Dbpedia* respectively.

Finally, we summarize the most important tradeoffs in Table 13.10. We represent the space ratio (in %) and the performance ratio (in each TP as well as the total mean) over the BTWO* proposal. We average over six datasets (described in Table 10.1): *2011 Australian Census*, *Jamendo*, *AEMET*, *Dbtune*, *2000 US Census* and *Dbpedia*. This table should be interpreted carefully. For instance, attending to these results, the BTWO*-OPS could be the candidate representation in most scenarios: it takes 87% the size of BTWO* and its performance is 0.8 times the corresponding in BTWO* (it performs 20% faster). However, we have described the resolution algorithms in detail, and we have shown that BTWO*-OPS strongly depends on the particular distribution of objects. In practice, we verified that it suffers from significant degradation in (s,p,V) and (s,V,V) whenever objects tend to be massively repeated (see Section 13.3.4): in *Dbpedia*, the resolution of (s,p,V) and (s,V,V) in BTWO*-OPS is 53% and 50% slower than BTWO* respectively, in contrast to the mean 1.0 values shown in the table.

Therefore, although these results can vary on specific datasets, Table 13.10 provides, though, a good indicator of the space/performance tradeoffs of the proposed triple indexes at consumer.

Gentlemen, you can't fight in here! This is the War Room.

Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1964)

14 Discussion

We briefly summarize the main contributions of this part devoted to RDF triple indexing (§14.1). We also depict potential uses besides `HDT` (§14.2).

14.1 Contributions

This part of the thesis is focused on scalability problems arising in RDF triple indexes for Big Semantic Data. Chapter 12 motivated this problem and provided a summary of the state of the art in RDF indexes and stores. We documented that most approaches suffer from scalability issues and use naive compression. With that in mind, we established the main goals of compact triple indexes on top of `HDT`-encoded datasets. These goals are addressed in Chapter 13, proposing lightweight indexes built efficiently at consumption time.

We employed succinct data structures for such indexes. First, we proposed a novel structure, referred to as Bitmap Triples (BT), which codifies the structure of the graph throw two correlated bitsequences. Its main advantage is that BT encoding can be enhanced (always at consumer) with a succinct index over the bitsequences, providing efficient (constant) SP-O access. Then, we introduced a Wavelet Tree which can replace the sequence of predicates, operating as a PS-O index in logarithmic time (*w.r.t* the number of predicates). Finally, an additional adjacency list of objects contributes with an OP-S indexing, completing the so-called BTWO* proposal for efficient RDF retrieval on top of `HDT`. One of the main contributions in this sense is that the resolution of all triple patterns in BTWO* was described algorithmically, and the costs were clearly detailed with the metrics proposed in Chapter 4.

We also presented a *GMR*-based alternative for the OP-S index. This configuration, BTWO-GMR*, is aimed at obtaining a more compact representation at the cost of performance degradation.

In our tests, we experimented the compressibility and query performance of all indexes on a testbed of real-world datasets, reporting important remarks:

- BT is the most compressed configuration for the `HDT` triples: in the considered datasets, BT size is about 60% the size of Compact Triples (CT) and up to 50% the size of Plain Triples. In turn, `HDT` with BT is 10% smaller than `HDT` with CT, on average.

- The SP-O index is a little 8% overhead over the transferred triples representation, whereas a mean of 20% of space overhead is required to build the Wavelet Tree (`PS-O` index). In contrast, the `OP-S` index could represent a cost of around 84% space overhead.

- The final BTWO* configuration provides total TP resolution with a mean of 93% space overhead of the exchanged BT.

- In general, our approach BTWO* reports the best overall performance for RDF retrieval, in comparison with k^2-triples and RDF3X proposals.

- BTWO* reports the worst performance in (V,p,V) patterns in which we pay the logarithmic costs of accessing the Wavelet Tree.

- On average, the k^2-triples solution outperforms 7 times our BTWO* compression. In contrast, BTWO* performs most triples patterns several order of magnitudes faster, being a mean of 344.80 times faster than k^2-triples. The improved k^2-triples+ solution is 5 times more compressed than BTWO*, but 15.32 times slower in query performance.
- BTWO* uses 6 times less space than RDF3X, and performs a mean of 33 times faster.

In addition, we analyzed other ordering variants for the triples. In particular, we show that OPS ordering demands a mean of 10% less space than SPO, and it excels in those triple patterns providing a constant object. In addition, it is competitive in the rest of the queries, hence some scenarios could choose this ordering instead of the SPO by default. In turn, POS ordering is slightly less compact than SPO, it excels retrieving by predicate but suffers significant degradation in the rest of the queries.

Finally, we studied the BTWO-GMR* interesting space/performance tradeoffs on both SPO and OPS orderings. This latter configuration, BTWO-GMR*-OPS, constitutes our most compressed solution: it takes a mean of 70% the size of BTWO*, at the cost of doubling the TP resolution time.

14.2 Other Applications

It is clear than all proposed indexes on top of HDT are closely tight to its particular representation. In particular, all them require i) a dictionary+triples partitioning and ii) a bitmap triples configuration (or a similar representation separating the data streams, predicates and objects, from the structure). Nonetheless, one might well wonder if these indexes could work out of HDT or with other diverse purposes.

In fact, all the proposed indexes could potentially be used off-HDT as additional structures complementing other systems. We summarize below the applicability with respect to SPARQL resolution and we briefly devise some applications. The coverage of the queries is always reported with respect to the aforementioned empirical study of real-world SPARQL queries (Arias et al., 2011).

- BT* resolves the most used TP combinations. For instance, it covers the 89% of the TP combinations in the Dbpedia query logs. As BT* is much smaller than other solutions (such as RDF3X as seen in the experimentation in Section 13.3), one could perfectly substitute (or complement) part of the indexes of the other solutions with BT*.
- We have shown that the Wavelet Tree pays a logarithmic time and performs slower than other solutions resolving (V,p,V). Nonetheless, only 3.45% of the TPs in the *Dbpedia* query logs are of this type. In addition, if this TP is part of a SPARQL BGP, a query planner could probably tend to avoid its early resolution, as it can provide too many results. All this states that i) the Wavelet Tree construction could potentially be obviated, or ii) another variant of indexes can be raised. Alternatively, the POS ordering could be chosen in some scenarios prioritizing (V,p,V) resolution.
- Although the *O-Index* results in the most overloaded structure in HDT, its size can also compete with the indexes of other approaches. Thus, the *O-Index* could be integrated within other solutions, as it also provides constant time in object accessing. Its *GMR* alternative provides noticeable space savings in conjunction with an OPS ordering at the cost of performance degradation.
- Our study of the impact of alternative triples orderings, as well as the proposed *GMR*-based index, provides a full set of flexible configurations to exploit those space/performance tradeoffs required by particular solutions.
- The demonstrated scalability in size and performance makes out indexes good candidates to take part of hybrid stores in-memory/disk in order to minimize I/O transactions.

Part V

Querying HDT-encoded Datasets

An elegant solution for keeping track of reality. Was it your idea?.

Inception (2010)

15 HDT Focusing on Querying (HDT-FoQ)

As we have motivated in the previous chapters, an HDT-encoded dataset can be directly accessed once its components are loaded into the memory hierarchy. Part III and IV of this thesis provided compact dictionary and triples components both for exchanging, as well as enhanced triple indexes built at consumption time. Thus, the next step was obvious: the integration of both research branches into an integrated proposal for RDF consumption.

This part of the thesis simply presents this integration, the so-called HDT Focusing on Querying (HDT-FoQ). In plain words, HDT-FoQ is the result of post-processing HDT for RDF consumption (Martínez-Prieto, Arias, & Fernández, 2012).

This chapter briefly presents some minor remarks on this integration (§15.1), as most of the work involves the development of the components described in the previous parts of this thesis. After these remarks, we evaluate the Publication-Exchange-Consumption workflow using HDT and HDT-FoQ on a real-world setup (§15.2). We analyze the performance of each step as well as the overall process and the query resolution.

15.1 Towards an HDT-FoQ Engine

HDT-FoQ is built, at consumption, on top of the exchanged HDT and exploits the presented dictionary and triple indexes to allow exchanged RDF to be directly consumed at large scale.

Previous chapters have shown that both dictionary and triples can be tuned carefully by considering the volume of the datasets and the retrieval velocity needed by specific applications. Nonetheless, we provide in the following a set of general decisions for post-processing and querying.

15.1.1 HDT-FoQ Generation

HDT-FoQ starts out from the idea of exchanging HDT with the $\mathcal{D}_{comp}$ dictionary (see Chapter 10) and the Bitmap Triples (see Chapter 13). Then, at consumption time, two processes are performed:

- It loads $\mathcal{D}_{comp}$ into the memory structures required to be functional. That is, it retrieves the data of all the compressed dictionary partitions in $\mathcal{D}_{comp}$, and loads them in the appropriated succinct data structures. Note that the $\mathcal{D}_{comp}$ pointers (*ptr*) and the language and type indexes (*lang* and *dtype*) have to be incorporated.

- It builds the BTWO* enhanced triple indexes. First, the object structure in Bitmap Triples is scanned to build the *O-index*. Then, the Wavelet Tree is constructed, deleting the previous predicate stream in Bitmap Triples.

The result is a compact RDF representation optimized to be managed and queried in main memory.

15.1.2 HDT-FoQ Querying

HDT-FoQ infrastructure enables basic triple patterns to be resolved, in compressed space, at higher levels of the hierarchy of memory. Note that $\mathcal{D}_{comp}$ provides the lookup operations (locate and extract) whereas BTWO* efficiently performs ID-triples retrieval. The conjunction of both components leads to resolve all SPARQL triples patterns.

Although this kind of queries are massively used in practice (Arias et al., 2011), the SPARQL core is defined around the concept of Basic Graph Pattern (BGP) and its semantics to build conjunctions, disjunctions, and optional parts involving more than a single triple pattern. Thus, HDT-FoQ must provide more advanced query resolution to reach a full SPARQL coverage. At this moment, we focus on resolving conjunctive queries by using specific implementations of the well-known *merge* and *index* join algorithms (Ramakrishnan & Gehrke, 2000). Additional operations and optimizations are relegated to future work.

BGP resolution. Efficient BGP resolution relies on i) the performance achieved for individual triple pattern resolution, ii) the efficiency of the join algorithms, and iii) the optimization strategies used for triple pattern reordering within the BGP. Query optimization is orthogonal to RDF retrieval, thus HDT-FoQ could take advantage of any existing technique within the state of the art. In the following, we provide insights into efficient join implementations on top of HDT-FoQ triple pattern resolution.

Merge and Index joins can be directly resolved in HDT-FoQ. Merge join is used when the results of both triple patterns are sorted by the join variable. It is worth noting that triple pattern results are given in the order provided by the index used (see Table 13.4). If the results of one triple pattern are not sorted by the join variable, index join can always be performed. It first retrieves all results for the join variable in one triple pattern and replaces them in the other one.

In our HDT-FoQ implementation (evaluated in Section 15.2), we follow a simple approach and our algorithm always performs index join. To do so, we first resolve the less expensive pattern, in terms of the expected number of results, substituting the join variable by the obtained values, and continue with the rest of the TPs. As we show below, the BTWO* indexes allow to obtain an expected number of results efficiently:

- (i,v,v) or (i,j,v): we pre-process the triple pattern making use of findPredicate(i) Bitmap Triples functionality (see Section 13.1.2). The range of positions in S_p indicates the expected object results.

- (v,v,k) or (v,j,k): we pre-process the triple pattern by means of occsObj operation of the OP-S index (see Section 13.2.2). If the predicate is given, in (v,j,k), we restrict the number of expected results to the number of predicate-object pairs.

- (i,v,k): this is the less tight estimation, we estimate it as (i,v,v). Nonetheless, note that his estimation should be very close to the real value, if we consider the results of our structural metrics; the out- and in-degrees were comparable to their corresponding direct degrees, stating that if a subject and an object are related, only one predicate brings these nodes together, on average (see results in Section 4.3.4).

- (v,j,v): due to the high Wavelet Tree costs, we store an histogram with the number of triples for each predicate beforehand, accessing it when deciding for the less expensive pattern. The size of this histogram is depreciable.

As mentioned, there is room for other optimizations on top of HDT-FoQ, but the presented approach sets the basis of BGP resolution and, thus, full SPARQL support.

Dataset	Original size	`gzip`	`HDT`	`HDT+gzip`
2011 Australian Census	52	1.46	1.57	0.35
Jamendo	144	8.41	14.71	6.17
AEMET	726	18.67	47.50	9.50
Dbtune	9,566	1,074.79	662.94	259.73
2000 US Census	21,796	1,007.79	813.85	209.44
Dbpedia 3-8	63,053	5,049.22	6,792.55	2,767.91

Table 15.1: Compressed sizes (MB).

15.2 Experimental Evaluation

This section analyzes the Publication-Exchange-Consumption workflow. The setup is similar to the configuration presented in Chapter 7: the **data publisher** is implemented on a powerful computational configuration whereas the **consumer** is slightly more limited (see Section 7.4 for complete details). In addition, we consider here a third involved agent, the network:

- The **network** is regarded as an ideal communication channel for a fair comparison. It is considered free of errors and any other external interference. We assume a transmission speed of 2Mbyte/s.

We make use of the corpora we are employing in the rest of the thesis, described in Section 4.2. As usual, we report "user" times in all experiments. The `HDT-FoQ` prototype is also developed in C++, compiled using `g++-4.6.1 -O3 -m64` and is publicly available at `http://www.rdfhdt.org`.

We first analyze the impact of using `HDT` as a basis for publication, exchange and consumption within the studied workflow, and compare its performance with respect to traditional methods currently used in each process. Then, we focus on studying the performance of `HDT-FoQ` as the querying infrastructure for SPARQL: we measure response times for triple pattern and join resolution.

15.2.1 Analyzing the Publication-Exchange-Consumption Workflow

Our analysis always considers that the publication is a one-time process (performed only once), whereas exchanging and preprocessing costs are paid each time that any consumer retrieves the published dataset. The publication policy affects the size of the datasets and, thus, i) the time for exchange but also ii) the decompression time, as this should be the initial consumption step when traditional compression is used for publication. We analyze the use a gzip compression as it reported good compression ratios in our previous evaluation (see Section 7.4.1) while providing the best size/time tradeoff.

We assume that the publication process begins with the dataset already serialized. Thus, gzip-based publication only considers the compression time, whereas processes based on `HDT` comprise the times required for generating the `HDT` representation (always at publisher) and its subsequent gzip compression (to obtain higher compression ratios). For the `HDT` dictionary, we make use of the $\mathcal{D}^{(Q)}_{comp}$ configuration, optimized for querying (see Section 10.5). The triples are encoded in Bitmap Triples (see Chapter 13).

Table 15.2 shows the time used for publication in the data provider: `gzip` is the faster choice and largely outperforms the `HDT`-based publication. Nevertheless, remember that this process is only performed once, hence size is a more important factor due to its influence on the subsequent processes. The publication size is drawn in Table 15.1, showing that `HDT+gzip` is the best choice. It achieves highly-compressed representations. For instance, `HDT+gzip` takes 1.8 times less space than `gzip` for `Dbpedia` and less than 3 times averaging all datasets. This spatial improvement determines the subsequent exchange and decompression (for consumption) times as shown in Tables 15.3 and 15.4.

In turn, the combination of `HDT` and `gzip` excels in exchange due to its high compressibility. Its transmission costs are clearly smaller than the other alternatives, being a noticeable saving in the largest datasets: HDT+gzip saves 1141 seconds (19 minutes) downloading Dbpedia and 399 seconds (almost 7 minutes) for the 2000 US Census. Moreover, thanks to its compressibility, HDT+gzip is also more efficient at decompression than universal compression over plain RDF (see Table 15.3). Note that HDT (not gzipped) does not need decompression, hence the 0-second column in Table 15.3.

Thus, HDT-based publication and its subsequent compression arises as the most efficient choice for exchanging RDF within the Web of Data.

Dataset	gzip	HDT	HDT+gzip
2011 Australian Census	0.73	2.19	2.25
Jamendo	1.65	12.18	12.93
AEMET	5.66	47.79	49.81
Dbtune	142.33	512.12	536.88
2000 US Census	201.66	990.41	1,012.51
Dbpedia 3-8	861.61	7,209.55	7,521.63

Table 15.2: Publication times (seconds).

Dataset	gzip	HDT	HDT+gzip
2011 Australian Census	0.73	0.79	0.18
Jamendo	4.21	7.36	3.09
AEMET	9.34	23.75	4.75
Dbtune	537.39	331.49	129.87
2000 US Census	503.90	406.93	104.72
Dbpedia 3-8	2,524.61	3,396.28	1,383.96

Table 15.3: Exchange times (seconds).

Dataset	gzip	HDT	HDT+gzip
2011 Australian Census	0.18	0.00	0.01
Jamendo	0.52	0.00	0.13
AEMET	2.29	0.00	0.36
Dbtune	87.27	0.00	4.46
2000 US Census	165.70	0.00	4.94
Dbpedia 3-8	540.81	0.00	61.64

Table 15.4: Decompression times (seconds).

Dataset	Virtuoso	RDF3X	HDT-FoQ
2011 Australian Census	1.53	2.45	0.11
Jamendo	4.88	8.84	0.28
AEMET	18.98	33.65	0.87
Dbtune	324.46	846.57	16.27
2000 US Census	699.00	1,977.60	29.52
Dbpedia 3-8	12,900.00	10,712.00	54.83

Table 15.5: Indexing times (seconds).

Dataset	`gzip->RDF3x`	`gzip->Virtuoso`	`HDT+gzip->HDT-FoQ`
2011 Australian Census	3.36	2.44	0.29
Jamendo	13.56	9.60	3.50
AEMET	45.28	30.60	5.98
Dbtune	1,471.23	949.13	150.60
2000 US Census	2,647.20	1,368.60	139.18
Dbpedia 3-8	13,777.42	15,965.42	1,500.43

Table 15.6: Overall times for exchanging+decompressing+indexing (seconds).

The next step focuses on making the exchanged RDF datasets queryable for consumption. As stated, the traditional process relies on indexing the plain RDF through any RDF store. We test this approach with two systems: `Virtuoso 7` (relational solution) and `RDF3X` (multi-indexing solution). These solutions are reviewed in Section 12.2. We compare their performance against `HDT-FoQ`, which builds additional structures on the `HDT`-serialized datasets previously exchanged.

Table 15.5 compares these times. As can be seen, `HDT-FoQ` excels for all datasets: `HDT-FoQ` indexing time is at least one order of magnitude faster than that obtained for the other techniques. For *Dbpedia*, `HDT-FoQ` loads in less than a minute, whereas Virtuoso and RDF3X performs in the range of 3 hours. This demonstrates how `HDT-FoQ` leverages the binary `HDT` representation to efficiently create its additional indexes and make RDF quickly queryable. This fact also shows that we successfully reduce the computation required by the consumer to make queryable RDF obtained within the Web of Data.

Overall Performance. We analyze, in the following, the time of the overall process for a consumer. Note that the publication process is decoupled from this analysis because it is performed only once, and its cost is attributed to the data provider. Thus, we consider the times for exchanging and consumption. These times are shown in Table 15.6, which compares the time needed for a conventional implementation against the `HDT` driven approach. In the traditional approach, the RDF is exchanged in gzip, decompressed at consumption and indexing with RDF3X or Virtuoso. With `HDT`, we take `HDT`-gzipped datasets for exchanging and the subsequent fast decompression and `HDT-FoQ` generation at consumption.

As can be seen, this workflow is completed faster using the `HDT` driven approach. In particular, the `HDT` solution finishes the workflow a mean of 7 and 10 times faster, on average, than Virtuoso and RDF3X respectively. This states that the consumer can start using the data in a shorter time (7-10 times faster on average), but also with a more limited computational configuration.

15.2.2 `HDT-FoQ` in Consumption: Performance for SPARQL Querying

Once the consumer has at his disposal the `HDT-FoQ` infrastructure, we study the performance of our `HDT-FoQ` proposal as the basis for SPARQL querying. We first show the spatial needs of `HDT-FoQ` to be efficiently loaded in the consumer configuration. Then, we measure the performance of triple pattern resolution, expecting good results on the basis of the triple indexes and $\mathcal{D}_{comp}$ dictionary (see Chapters 13 and 10). Additionally, we test our basic join query resolution, presented in Section 15.1.2. Our main goal is to show the `HDT-FoQ` efficiency for RDF retrieval, but also to envision the potential for joins and thus to demonstrate its capabilities for SPARQL resolution on top of `HDT-FoQ`. We compare our results with respect to the indexing systems presented above, Virtuoso and RDF3X.

Table 15.7 summarizes the sizes of the indexes of each studied solution. The rightmost columns, `HDT` and `HDT-FoQ` respectively, show the size of the original `HDT` representation (after decompression) and the resultant in-memory configuration built on top of it. It is worth remembering that the figures reported for `HDT-FoQ` also include the overhead required for managing it in main memory. In turn, we emphasize that the sizes reported for `RDF3X` and `Virtuoso` are in-disk figures.

Dataset	Original Size (MB)	Virtuoso	RDF3X	`HDT`	`HDT-FoQ`
2011 Australian Census	52	38.46%	27.91%	3.02%	4.96%
Jamendo	144	109.72%	66.15%	10.22%	12.31%
AEMET	726	65.33%	33.67%	6.55%	8.07%
Dbtune	9,566	41.48%	43.13%	6.93%	9.16%
2000 US Census	21,796	25.40%	30.32%	3.73%	6.39%
Dbpedia 3-8	63,053	73.91%	48.46%	10.77%	13.36%
MEAN	-	59.05%	41.61%	6.87%	9.04%

Table 15.7: Indexing sizes (% over the original).

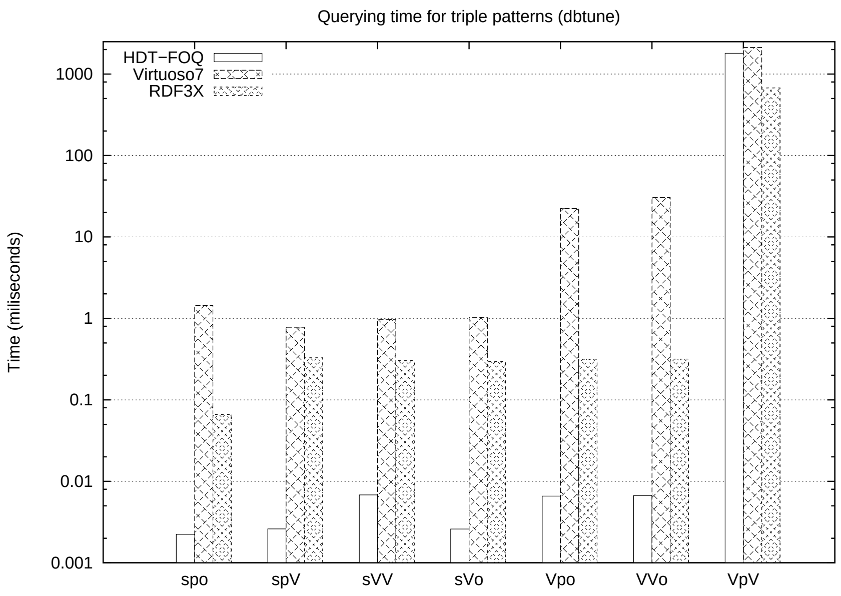


Figure 15.1: `HDT-FOQ` TP query performance in *Dbtune*.

These results place `HDT-FoQ` as the most compact index in this evaluation. Note that we have shown, in the evaluation in Section 13.3.3, that k^2-triples was the most compressed solution for RDF triple indexing. However, it performed significant slower than `HDT-FoQ`, and it lacks of a functional dictionary, hence it was not a potential candidate, at this moment, for SPARQL evaluation.

As can be seen in Table 15.7, `HDT-FoQ` takes a mean of 39% of extra space on top of `HDT` representations, and around 9% the original size of the dataset (in N-Triples). In summary, one could see that `HDT-FoQ` excels in size: the consumer can manage more than 431 million triples (in *Dbpedia*) using `HDT-FoQ`, sizing slightly more than 8GB in memory.

Finally, query performance is evaluated over *Dbtune* and the *2000 US Census*. For each one, we design a testbed of randomly generated queries which covers the entire spectrum of triple patterns and joins. We consider 5000 random triple patterns of each type (`(?S,P,?O)` is limited by the number of different predicates). To test conjunctive queries, we split joins into Subject-Subject (`SS`), Object-Object (`OO`) and Subject-Object (`SO`) categories. These represent the most used variants in which the variable of the join appears (Arias et al., 2011). For each category, we generate 1000 random queries with the appropriate constant values in the non-join positions. The join variable is the selected projection, *i.e.* the expected result of the query.

Querying times are obtained by running 5 independent executions of the testbed and averaging total user times. We compare `HDT-FoQ` against RDF3X and Virtuoso in a warm scenario (we run 5 previous

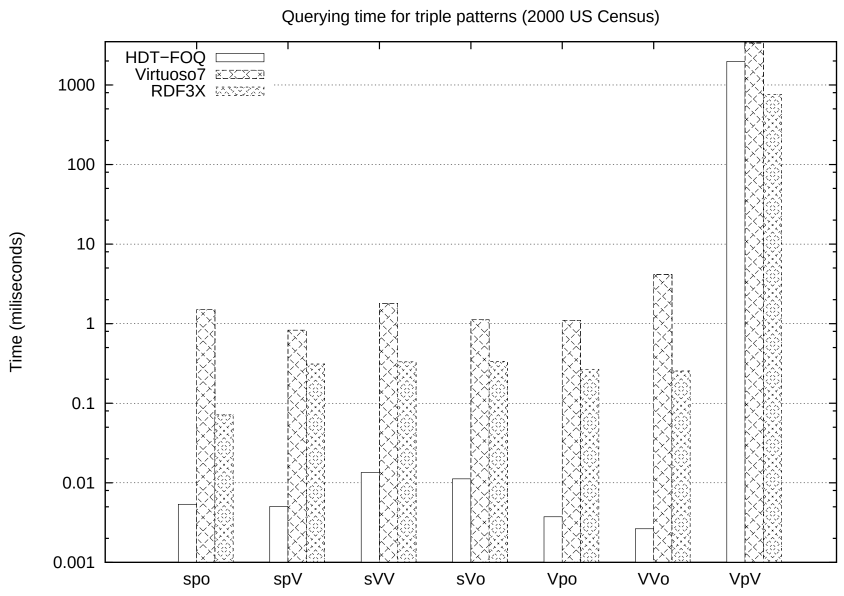


Figure 15.2: `HDT-FOQ` TP query performance in the *2000 US Census*.

executions before measuring time).

Figures 15.1 and 15.2 show the triple pattern resolution time in *Dbtune* and the *2000 US Census* respectively. It is worth noting that `HDT-FoQ` excels for almost every individual triple pattern. It speeds-up their resolution up to 2 orders of magnitude, only losing performance in (V,p,V), in which a logarithmic cost is paid for accessing predicates in the Wavelet Tree. Even in such case, only RDF3X performs faster than `HDT-FoQ` in both datasets.

The analysis of join performance is based on the results reported in Figures 15.3 and 15.4. These results show that i) `HDT-FoQ` is faster than RDF3X for the three considered categories of two-ways joins, in both datasets. In fact, it clearly outperforms RDF3X resolution, being a mean of 8 times faster. This difference is slightly reduced in SS joins: as the join variable is in the subject position, all accesses are by object, and thus the potential walks over the Wavelet Tree are penalized. In contrast, ii) `HDT-FoQ` is slower than Virtuoso (version 7) in joins. Although `HDT-FoQ` is the most efficient choice for triple pattern resolution, the join queries in Virtuoso are clearly optimized. Nonetheless, optimized join algorithms implemented on top of `HDT-FoQ` would allow it to compete fairly in this latter case by leveraging `HDT-FoQ` performance for triple pattern resolution.

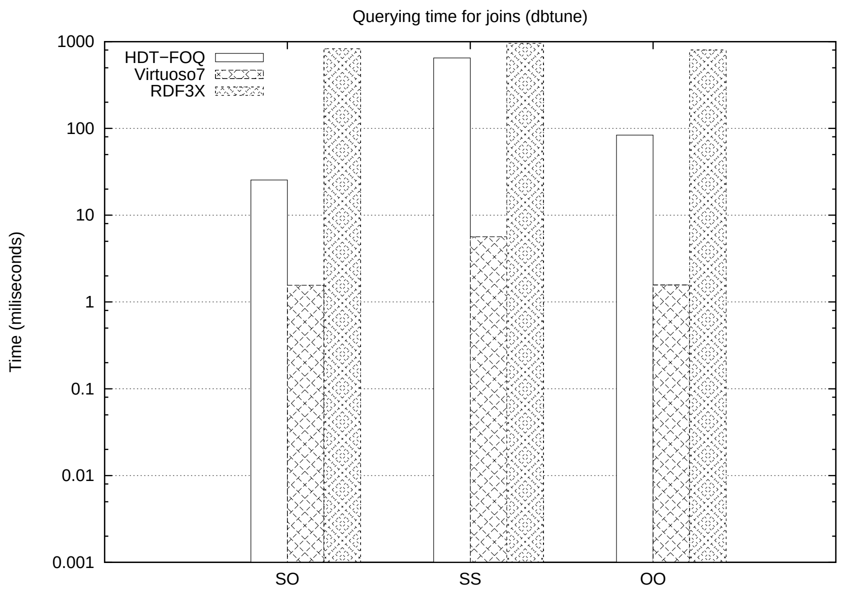


Figure 15.3: HDT-FOQ join performance in *Dbtune*.

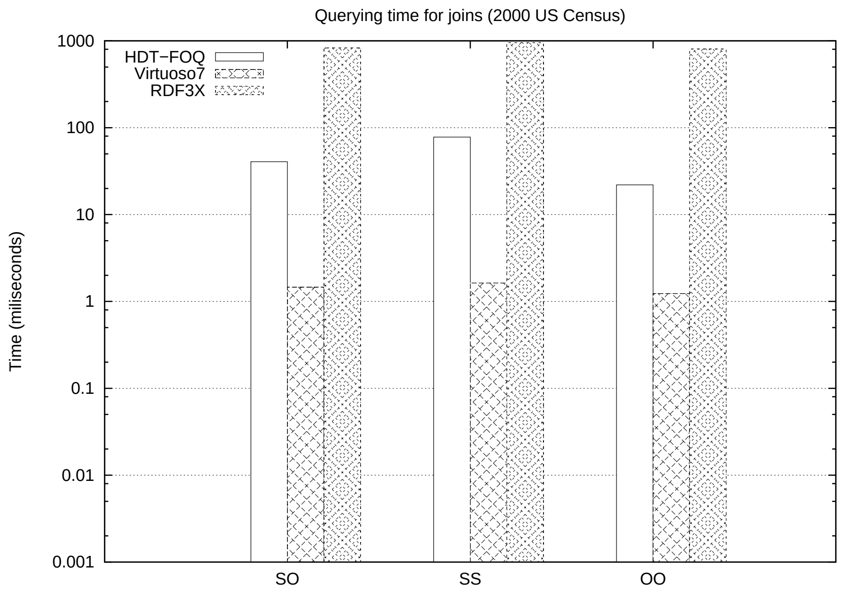


Figure 15.4: HDT-FOQ join performance in the *2000 US Census*.

Part VI

Thesis Summary

Maybe some day you forget what it's like to be human and maybe then, it's ok.

Mystic River (2003)

16 Conclusions and Future Work

This chapter concludes summarizing the most important contributions of this thesis (§16.1) as well as devising future work (§16.2). Note that detailed discussions are also provided in the final chapters of each part of the thesis (Chapters 5, 8, 11 and 14).

16.1 Summary of Contributions

This thesis presents basic foundations for Big Semantic Data management. First, we trace a route from the current data deluge, the concept of Big Data and the need of machine-processable semantics on the WWW. The Resource Description Framework (RDF) and the Web of (Linked) Data naturally emerge in this well-grounded scenario. The former, RDF, is the natural data model for semantic data, combining the flexibility of semantic networks with a graph data structure that makes it an excellent choice for describing metadata at Web Scale. The latter, the Web of (Linked) Data, provides a set of rules to publish and link Big Semantic Data.

Nonetheless, the Web of Data suffers from diverse scalability problems when moving to a RDF data-intense processing era. We justify the different and various management problems arising in Big Semantic Data by characterizing their main stakeholder. Then, we define a common workflow *Publication-Exchange-Consumption*, existing in most applications in the Web of Data. Traditional verbose RDF formats remain as one of the main bottlenecks at exchanging and post-processing. Inherent scalability drawbacks of huge RDF graphs discourage their consumption due to the space they take up, the powerful resources and the large time required to process them.

This thesis addresses these problems i) studying the underlying RDF structure essence, ii) proposing a novel RDF binary format (`HDT`) iii) giving compact RDF dictionaries and iv) succinct triple structures which can be efficiently serialized for exchanging and can be enhanced with additional indexes to be queried at consumption without decompression.

We propose and define novel metrics characterizing real-world RDF data. We provide a toolkit of parameters determining common and particular features in RDF modeling. These metrics are used in the thesis to finely parametrize our proposed indexes. We hope they become a useful handbook when developing or optimizing any kind of semantic data structures.

The scalability problems arising to the current state-of-the-art management solutions within this scenario set the basis of our integrated proposal `HDT`. `HDT` is designed as a binary RDF format to fulfill the requirements of portability (from and to other formats), compact ability, parsing efficiency (readiness for post-processing) and direct access to any piece of data in the dataset. We detail the design of `HDT` components (Header, Dictionary and Triples), their different operations and intended use. We also instantiate a concrete practical deployment of `HDT` for publication and exchanging and we develop an `RDF/HDT` syntax specification.

Next, we focus on optimizing both dictionary and triples components, with efficient consumption in mind. We first address compressed representations for RDF dictionaries, adapting existing techniques for compressed string dictionaries. The proposed solution, a novel RDF dictionary called $\mathcal{D}_{comp}$, achieves the best compression ratios in the experimentation. Besides, its space/time can be finely tuned, outperforming the lookup performance of traditional approaches. Moreover, the organization of subdictionaries in $\mathcal{D}_{comp}$ and its *regex* resolution features open up further optimizations for filter resolution.

Regarding the RDF structure encoded in the triples component, we focus on boosting its navegability and ulterior consumption processes. To do so, we first propose a novel triple organization and encoding called Bitmap Triples: it sees the graph as a forest of trees and codifies its structure in two correlated bitsequences. This decision improves size but, more important, allows succinct data structures to operate in the encoded structure.

We argue that HDT-encoded datasets can be directly consumed within the presented workflow. Thus, we show that novel indexes, on the basis of succinct data structures, can be created once the different components are loaded into the memory hierarchy at the consumer. This allows to provide a compressed, in-memory solution which resolves all kind of SPARQL triple patterns. Moreover, the final configuration of triple indexes at consumer, called BTWO*, is perfectly described algorithmically, and the costs were clearly detailed with the metrics proposed. Our experimentation shows that, in general, our approach BTWO* reports the best overall performance for RDF retrieval. We also present different variants (in triples ordering and alternative indexes) to provide a complete set of configurations exploiting space/performance tradeoffs.

Finally, we integrate the $\mathcal{D}_{comp}$ dictionary in the core of HDT-based solutions, and we consider the creation of the BTWO* indexes at consumption. This compact infrastructure, called HDT-FoQ (HDT Focused on Querying) is evaluated toward the traditional combination of universal compression (for exchanging) and RDF indexing of the plain RDF (for consumption).

Experiments show how HDT excels at almost every stage of the Publication-Exchange-Consumption workflow. Experiments reports that the publisher could spend a bit more time to encode the Big Semantic dataset in HDT, but in return, this hugely favours the consumption; consumer is able to exchange it three times faster (on average), and, more important, the indexing time is largely reduced to just a few seconds for huge datasets with a limited configuration of resources. Therefore, the time since a machine or human client discovers the dataset until she is ready to start querying its content is reduced up to 19 times by using HDT instead of the traditional approaches (8.66 times on average). Furthermore, the query performance is very competitive compared to state-of-the-art RDF stores; the aggressive size reduction allows to operate a vast amount of triples in main memory, avoiding slow I/O transferences. HDT-FoQ excels in triple pattern resolution and remain competitive in basic join resolution, setting the base of an HDT-based store serving SPARQL.

In short, HDT-based solutions arises as the most efficient choice for publication and exchange of Big Semantic Data, and set the basis of optimal consumption in the Web of Data.

16.2 Future Work

These results open up interesting issues for future work. We should work on improving predicate-based retrieval because it reports the less-competitive performance. Our on-going work relies on the optimization of the predicate index by tuning the trade-off between access time and spatial needs. In addition, we plan to optimize our join algorithms with *Sideways Information Passing* (SIP) mechanisms, proposed by Neumann and Weikum (2009). SIP is about passing on-the-fly information between both TPs, hence the join is interactively evaluated without materialization of intermediate results. We believe that our efficient resolution of TPs as well as early cardinality estimations can perfectly fit the SIP mechanism.

In parallel, there are several areas where HDT can be further exploited. We foresee a huge potential of HDT to support many aspects of the workflow Publication-Exchange-Consumption. HDT-based technologies can emerge to provide supporting tools for both publishers and consumers. For instance a very useful tool for a publisher is setting up a SPARQL endpoint on top of an HDT file. As the experiments show, HDT-FoQ is very competitive on queries, but there is still plenty of room for SPARQL optimization, by leveraging efficient resolution of triple patterns, joins and query planning. Another useful tool for publishers is configuring a dereferenceable URI materialization from a given `HDT`. Here the experiments also show that performance will be very high because `HDT-FoQ` is really fast on queries with a fixed RDF subject.

Finally, although the use of succinct data structures allows more data to be managed in the main memory, it could still remain excessive for consumers with limited memory. Under this scenario, we devise an evolution of `HDT-FoQ` to perform as an in-memory/on-disk system providing dynamic data management, *i.e.*, efficient insertion, updating and deletion of triples at consumption. In this sense, we works on a particular architecture for Big Semantic Data management in real-time. Our initial proposal is called SOLID (Cuesta, Martínez-Prieto, & Fernández, 2013). This tiered architecture separates the complexities of Big Semantic Data management from their real-time data generation and consumption. Whereas the Big Semantic Data can be stored following `HDT` and indexed as `HDT-FoQ`, the dynamics of real-time are addressed using NoSQL technology. Two additional layers are required to integrate both worlds i) when resolving questions, as both novel and historic data should be queried and their results have to be integrated and ii) when merging, at a given moment, the real-time data with the `HDT` information (`HDT-FoQ` indexes must be rebuilt as well). We hope this architecture to fully accomplish the requirements of Big Semantic management in most practical scenarios.

All right. This one time I'll let you ask me about my affairs.

The Godfather (1972)

A Publications and other Results

This chapter summarizes the publications of the author directly related with this thesis[1]. We finally include the research stays during the research period.

Selected Publications

ISI-Ranked Journals

- 2013
  - Javier D. Fernández, Miguel A. Martínez-Prieto, Claudio Gutiérrez, Axel Polleres, Mario Arias Gallego. Binary RDF Representation for Publication and Exchange (HDT). *Journal of Web Semantics*,19:22-41, Elsevier, 2013. ISSN 1570-8268.

    This article has been cited by:

    (1) Hagedorn, S., Sattler, K. U. Efficient Parallel Processing of Analytical Queries on Linked Data. In *On the Move to Meaningful Internet Systems: OTM 2013 Conferences*, pp. 452-469. Springer Berlin Heidelberg, 2013.

    (2) Zimmermann, A., Gravier, C., Subercaze, J., Cruzille, Q. Nell2RDF: Read the Web, and turn it into RDF. In 2^{nd} *International Workshop on Knowledge Discovery and Data Mining Meets Linked Open Data* (Know@ LOD), 2013.

    (3) Kasten, A., Scherp, A. Iterative signing of RDF(S) graphs, named graphs, and OWL graphs: Formalization and application. Technical report Nr. 3/2013, University of Koblenz-Landau, 2013. Available at: `http://uni-koblenz.de/~fb4reports/2013/2013_03_Arbeitsberichte.pdf`, retrieved October 2013.
  - (PR) Gustavo A. Pabón, Claudio Gutiérrez, Javier D. Fernández, Miguel A. Martínez-Prieto. Publication of Linked Open Census Microdata. *Journal of the American Society for Information Science and Technology*, 64 (9):1802-1814, ASIS&T, 2013. ISSN 1532-2890.

Other Journals

- 2012
  - Miguel A. Martínez-Prieto, Javier D. Fernández, Rodrigo Cánovas. Querying RDF Dictionaries in Compressed Space. *ACM SIGAPP Applied Computing Review*, 12(2): 64-77, ACM, 2012. ISSN 1559-6915.
  - Javier D. Fernández, Miguel A. Martínez-Prieto, Mario Arias. Scalable Management of Compressed Semantic Big Data. *ERCIM News*, 89: 29-30, ERCIM, 2012. ISSN 0926-4981.

[1]Those articles partially related with this thesis are marked as **PR**: Partially Related.

This article has been cited by:

(1) Adamou, A. An architecture for scaling ontology networks. *Doctoral dissertation*, Università di Bologna, 2013. Available at `http://amsdottorato.cib.unibo.it/5528/`, retrieved October 2013.

Chapters in Books

- 2013
  - Javier D. Fernández, Mario Arias, Miguel A. Martínez-Prieto, Claudio Gutiérrez. Management of Big Semantic Data. Akerkar, Rajendra (Ed.): Big Data Computing, Taylor and Francis/CRC, ISBN: 978-1-46-657837-1.

Standards

- 2011
  - Javier D. Fernández, Miguel A. Martínez-Prieto, Claudio Gutiérrez, Axel Polleres. Binary RDF Representation for Publication and Exchange (HDT). *W3C Member Submission*, March 30, 2011, `http://www.w3.org/Submission/2011/03/`.

    This standard has been cited by:

    (1) Thoma, M., Antonescu, A. F., Mintsi, T., Braun, T. Linked Services for M2M communication with Enterprise IT systems. In 9th *International Wireless Communications and Mobile Computing Conference* (IWCMC), pp. 1212-1216, 2013.

    (2) Kaoudi, Z., Koubarakis, M. Distributed RDFS Reasoning Over Structured Overlay Networks. *Journal on Data Semantics*, pp. 1-39, 2012

    (3) Barnaghi, P., Wang, W., Henson, C., Taylor, K. Semantics for the Internet of Things: Early Progress and Back to the Future. *International Journal on Semantic Web and Information Systems* (IJSWIS), 8(1): 1-21, 2012.

International Conferences and Workshops

- 2013
  - (PR) Carlos E. Cuesta, Miguel A. Martínez-Prieto, Javier D. Fernández Towards an Architecture for Managing Big Semantic Data in Real-Time. In 7th *European Conference on Software Architecture (ECSA)*, pp. 45-53, LNCS 7957, Springer-Verlag, 2013.
- 2012
  - Miguel A. Martínez-Prieto, Javier D. Fernández, Rodrigo Cánovas. Compression of RDF Dictionaries. In 27th *ACM International Symposium on Applied Computing (SAC 2012) - Track The Semantic Web and Applications (SWA)*, pp. 340-347, ACM Press, 2012.

    This paper has been cited by:

    (1) Grund, M., Cudre-Mauroux, P., Krueger, J., Plattner, H. Hybrid graph and relational query processing in main memory. In 29th *International Conference on Data Engineering Workshops* (ICDEW), pp. 23-24, 2013.

- Miguel A. Martínez-Prieto, Mario Arias, Javier D. Fernández. Exchange and Consumption of Huge RDF Data. In 9th Extended Semantic Web Conference (ESWC), pp. 437-452, LNCS 7295, Springer-Verlag, 2012.

  This paper has been cited by:

  (1) Ashraf, J. A semantic framework for ontology usage analysis. Ph.D. Curtin University, School of Information Systems, Curtin Business School, 2013. Available at: `http://trove.nla.gov.au/work/183091728?q&versionId=199455331`, retrieved October 2013.

- Javier D. Fernández. Binary RDF for Scalable Publishing, Exchanging and Consumption in the Web of Data. In 21st *International World Wide Web Conference* (WWW), pp. 133-138, 2012.

  This paper has been cited by:

  (1) Horridge, M., Redmond, T., Tudorache, T., Musen, M. Binary OWL. In 10th *OWL: Experiences and Directions Workshop* (OWLED 2013), 2013. Available at: `http://webont.org/owled/2013/papers/owled2013_12.pdf`, retrieved October 2013.

• 2011

- Mario Arias, Javier D. Fernández, Miguel A. Martínez-Prieto, Claudio Gutierrez. HDT-it: Storing, Sharing and Visualizing Huge RDF Datasets. In 10th *International Semantic Web Conference (ISWC 2011)*, track: poster session, 2011. Available at: `http://dataweb.infor.uva.es/wp-content/uploads/2011/10/iswc2011.pdf`, retrieved October 2013.
- Sandra Álvarez, Nieves R. Brisaboa, Javier D. Fernández, Miguel A. Martínez-Prieto. Compressed k^2-Triples for Full-In-Memory RDF Engines. In 17th *Americas Conference on Information Systems (AMCIS 2011)*: article 350, 2011.

  This paper has been cited by:

  (1) Brisaboa, N. R., Ladra, S., Navarro, G. Compact representation of Web graphs with extended functionality. *Information Systems*, 39(1):152-174, 2014.
  (2) Joshi, A. K., Hitzler, P., Dong, G. Logical Linked Data Compression. In *The Semantic Web: Semantics and Big Data*, pp. 170-184, Springer Berlin Heidelberg, 2013.
  (3) Tran, T., Ladwig, G., Rudolph, S. Managing Structured and Semistructured RDF Data Using Structure Indexes, *IEEE Transactions on Knowledge and Data Engineering*, 25(9): 2076-2089, 2013.
  (4) Brisaboa, N. R., de Bernardo, G., Navarro, G. Compressed Dynamic Binary Relations. In *Data Compression Conference* (DCC), pp. 52-61, 2012.
  (5) Joshi, A. K., Hitzler, P., Dong, G. Towards Logical Linked Data Compression. In the *Joint Workshop on Large and Heterogeneous Data and Quantitative Formalization in the Semantic Web* (LHD+ SemQuant), 2012.

- Javier D. Fernández, Miguel A. Martínez-Prieto, Claudio Gutiérrez. Publishing Open Statistical Data: the Spanish Census. In 12th *Annual International Conference on Digital Government Research (dg.o 2011)*, pp. 20-25, 2011. **Best Policy Paper Award.**

This paper has been cited by:

(1) Meroño-Peñuela, A., Guéret, C., Hoekstra, R., Schlobach, S. Detecting and Reporting Extensional Concept Drift in Statistical Linked Data. In 1^{st} *International Workshop on Semantic Statistics* (SemStats 2013), 2013. Available at: `http://www.cedar-project.nl/wp-content/uploads/semstats2013_submission_7.pdf`, retrieved October 2013.

(2) Otjacques, B., Stefas, M., Cornil, M., Feltz, F. Open data visualization keeping traces of the exploration process. In 1^{st} *International Workshop on Open Data*, pp. 53-60, 2012.

– Mario Arias Gallego, Javier D. Fernández, Miguel A. Martínez-Prieto, Pablo de la Fuente. RDF Visualization using a Three-Dimensional Adjacency Matrix. In 4^{th} *International Semantic Search Workshop (SemSearch 2011)*, 2011. Available at: `http://km.aifb.kit.edu/ws/semsearch11/8.pdf`, retrieved October 2013.

This paper has been cited by:

(1) Gottron, T., Pickhardt, R. A detailed analysis of the quality of stream-based schema construction on linked open data. In *Semantic Web and Web Science*, pp. 89-102. Springer New York. 2013

– Javier D. Fernández. HDT: Logical RDF Partitioning for Publishing and Exchanging in the Web of Data. In 5^{th} *Alberto Mendelzon Workshop (AMW 2011)*, Student Papers, 2011. Available at `http://dataweb.infor.uva.es/amw2011_submission_16.pdf`, retrieved October 2013.

– Mario Arias Gallego, Javier D. Fernández, Miguel A. Martínez-Prieto, Pablo de la Fuente. An Empirical Study of Real-World SPARQL Queries. In 1^{st} *International Workshop on Usage Analysis and the Web of Data* (USEWOD 2011), 2011. **Best Challenge Paper Award.** Available at: `http://arxiv.org/abs/1103.5043`, retrieved October 2013.

This paper has been cited by:

(1) Atre, M. OptBitMat: For SPARQL OPTIONAL (left-outer-join) queries. In *arXiv preprint*, 2013. Available at `http://arxiv.org/pdf/1304.7799v2`, retrieved October 2013.

(2) Buil-Aranda, C., Hogan, A., Umbrich, J., Vandenbussche, P. Y. SPARQL Web-Querying Infrastructure: Ready for Action? In *12th International Semantic Web Conference* (ISWC), to be published, 2013. Available at: `http://www.deri.ie/sites/default/files/publications/paperiswc.pdf`, retrieved October 2013.

(3) Rietvelda, L., Hoekstraa, R. YASGUI: How do we Access Linked Data? In *Semantic Web journal*, in review, 2012. Available at: `http://www.semantic-web-journal.net/system/files/swj538.pdf`, retrieved October 2013.

(4) Shekarpour, E. M. S., Auer, S., Ngomo, A. C. N. Large-scale RDF Dataset Slicing. In 7^{th} *IEEE International Conference on Semantic Computing* (ICSC 2013), to be published, 2013. Available at: `https://bitbucket.org/emarx/rdfslice/downloads/slice_v1.2.pdf`, retrieved October 2013

(5) Urbani, J. On Web-scale Reasoning. *Doctoral dissertation*, Amsterdam: Vrije Universiteit, 2013. Available at: `http://www.cs.vu.nl/~bal/dissertation-Urbani.pdf`, retrieved October 2013.

(6) De Saint-Marcq, V. L. C., Deville, Y., Solnon, C., Champin, P. A. Un solveur léger efficace pour interroger le Web Sémantique. In *Huitièmes Journées Francophones de Programmation par Contraintes* (JFPC), (in French), 2012. Available at: `http://hal.inria.fr/docs/00/80/98/59/PDF/jfpc2012.pdf`, retrieved October 2013.

(7) De Virgilio, R. A linear algebra technique for (de) centralized processing of SPARQL queries. In *Conceptual Modeling*, pp. 463-476, Springer Berlin Heidelberg, 2102.

(8) Elbedweihy, K., Wrigley, S. N., Ciravegna, F. Improving Semantic Search Using Query Log Analysis. In *Workshop on Interacting with Linked Data* (ILD 2012), pp. 61-74, 2012.

(9) Karnstedt, M., Sattler, K. U., Hauswirth, M. Scalable distributed indexing and query processing over Linked Data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 10:3-32, 2012.

(10) Kotoulas, S., Urbani, J., Boncz, P., Mika, P. Robust runtime optimization and skew-resistant execution of analytical SPARQL queries on pig. In *The Semantic Web-ISWC 2012*, pp. 247-262, Springer Berlin Heidelberg, 2012.

(11) Letelier, A., Pérez, J., Pichler, R., Skritek, S. Static analysis and optimization of semantic web queries. In 31^{st} *symposium on Principles of Database Systems*, pp. 89-100, 2012. Görlitz, O., Thimm, M., Staab, S. SPLODGE: systematic generation of SPARQL benchmark queries for linked open data. In *The Semantic Web-ISWC 2012*, pp. 116-132, Springer Berlin Heidelberg, 2012.

(12) Picalausa, F., Luo, Y., Fletcher, G. H., Hidders, J., Vansummeren, S. A structural approach to indexing triples. In *The Semantic Web: Research and Applications*. pp. 406-421, Springer Berlin Heidelberg, 2012.

(13) Prasser, F., Kemper, A., Kuhn, K. A. Efficient distributed query processing for autonomous RDF databases. In 15^{th} *International Conference on Extending Database Technology*, pp. 372-383, 2012.

(14) Raghuveer, A. Characterizing Machine Agent Behavior through SPARQL Query Mining. In 2^{nd} *International Workshop on Usage Analysis and the Web of Data* (USEWOD 2012), 2012. Available at: `http://ir.ii.uam.es/usewod2012/usewod2012_raghuveer.pdf`, retrieved October 2013.

(15) Umbrich, J. A Hybrid Framework for Querying Linked Data Dynamically, *Doctoral dissertation*, National University of Ireland, Galway, 2012. Available at: `http://hdl.handle.net/10379/3360`, retrieved October 2013.

(16) Umbrich, J., Hogan, A., Polleres, A., Decker, S. On Link Traversal Querying for a diverse Web of Data. In *Semantic Web journal*, in review, 2012. Available at: `http://www.semantic-web-journal.net/system/files/swj318_0.pdf`, retrieved October 2013.

(17) Berendt, B., Hollink, L., Hollink, V., Luczak-Rösch, M., Möller, K., Vallet, D. Usage analysis and the web of data. In *ACM SIGIR Forum*, 45(1), pp. 63-69, ACM, 2011.

(18) Elbedweihy, K., Mazumdar, S., Cano, A. E., Wrigley, S. N., Ciravegna, F. Identifying Information Needs by Modelling Collective Query Patterns. In 2^{nd} *International Workshop on Consuming Linked Data* (COLD2011), 2011. Available at: `http://ceur-ws.org/Vol-782/ElbedweihyEtAl_COLD2011.pdf`, retrieved October 2013.

(19) Ell, B., Vrandecic, D., Simperl, E. Deriving human-readable labels from SPARQL queries. In 7^{th} *International Conference on Semantic Systems*, pp. 126-133, ACM, 2011.

(20) Lin, F., Krizhanovsky, A. Multilingual ontology matching based on Wiktionary data accessible via SPARQL endpoint. In 13^{th} *All-Russian Scientific Conference "Digital libraries: Advanced Methods and Technologies, Digital Collections"* (RCDL 2011), pp. 1-8, 2011.

(21) Mazumdar, S., Elbedweihy, K., Cano, A. E., Wrigley, S. N., Ciravegna, F. SEMLEX-A Framework for Visually Exploring Semantic Query Log Analysis. In 10^{th} *International Semantic Web Conference* (ISWC), (Demo), 2011. Available at: `http://iswc2011.semanticweb.org/fileadmin/iswc/Papers/PostersDemos/iswc11pd_submission_87.pdf`, retrieved October 2013.

(22) de Saint-Marcq, V. L. C., Deville, Y., Solnon, C. An efficient light solver for querying the semantic web. In *Principles and Practice of Constraint Programming* (CP 2011), pp. 145-159, Springer Berlin Heidelberg, 2011.

(23) Williams, G. T., Weaver, J. Enabling fine-grained HTTP caching of SPARQL query results. In *The Semantic Web-ISWC 2011*,pp. 762-777. Springer Berlin Heidelberg, 2011.

(24) Picalausa, F., Vansummeren, S. What are real SPARQL queries like? In 3rd *International Workshop on Semantic Web Information Management*, article 7, 2011.

- Javier D. Fernández. DataWeb: Compression, Indexing and Applications on Large Datasets. In 7th *Reasoning Web Summer School (RW 2011)*, 2011. **Best Poster Award.**

• 2010

- Javier D. Fernández, Claudio Gutiérrez, Miguel A. Martínez-Prieto. RDF Compression: Basic Approaches. In 19th *International World Wide Web Conference (WWW 2010)*, pp. 1091-1092, ACM Press, 2010.

  This paper has been cited by:

(1) Joshi, A. K., Hitzler, P., Dong, G. Logical Linked Data Compression. In *The Semantic Web: Semantics and Big Data*, pp. 170-184, Springer Berlin Heidelberg, 2013.

(2) Urbani, J. On Web-scale Reasoning. *Doctoral dissertation*, Amsterdam: Vrije Universiteit, 2013. Available at: `http://www.cs.vu.nl/~bal/dissertation-Urbani.pdf`, retrieved October 2013.

(3) Urbani, J., Maassen, J., Drost, N., Seinstra, F., Bal, H. Scalable RDF data compression with MapReduce. In *Concurrency and Computation: Practice and Experience*, 25(1): 24-39, 2013.

(4) Hogan, A., Umbrich, J., Harth, A., Cyganiak, R., Polleres, A., Decker, S. An empirical survey of Linked Data conformance. In *Web Semantics: Science, Services and Agents on the World Wide Web*, 14:14-44, 2012.

(5) Hogan, A., Zimmermann, A., Umbrich, J., Polleres, A., Decker, S. Scalable and Distributed Methods for Resolving, Consolidating, Matching and Disambiguating Entities in Linked Data Corpora. In *Journal of Web Semantics*, 10: 76-110, 2012.

(6) Joshi, A. K., Hitzler, P., Dong, G. Towards Logical Linked Data Compression. In the *Joint Workshop on Large and Heterogeneous Data and Quantitative Formalization in the Semantic Web* (LHD+ SemQuant), 2012.

(7) Hogan, A. Exploiting RDFS and OWL for Integrating Heterogeneous, Large-Scale, Linked Data Corpora. *Doctoral dissertation*, National University of Ireland, Galway, 2011. Available at: `http://sw.deri.org/~aidanh/docs/thesis/thesis-one-sided.pdf`, retrieved October 2013)

- (PR) Miguel A. Martínez-Prieto, Joaquín Adiego, Pablo de la Fuente, Javier D. Fernández. High-Order Text Compression on Hierarchical Edge-Guided. In *Data Compression Conference (DCC 2010)*, p.543, IEEE Computer Society Press, 2010.

- Javier D. Fernández, Miguel A. Martínez-Prieto, Claudio Gutiérrez. Compact Representation of Large RDF Data Sets for publishing and exchange. In 9th *International Semantic Web Conference (ISWC 2010)*, LNCS 6496, pp. 193-208, Springer-Verlag, 2010.

This paper has been cited by:

(1) Joshi, A. K., Hitzler, P., Dong, G. Logical Linked Data Compression. In *The Semantic Web: Semantics and Big Data*, pp. 170-184, Springer Berlin Heidelberg, 2013.

(2) Kunze, S. R., Auer, S. Dataset Retrieval. In 7^{th} *IEEE International Conference on Semantic Computing* (ICSC 2013), to be published, 2013.

(3) Tzitzikas, Y., Kampouraki, M., Analyti, A. Curating the Specificity of Ontological Descriptions under Ontology Evolution. In *Journal on Data Semantics*, pp. 1-32, 2013.

(4) Chekol, M. W. Analyse Statique de Requête pour le Web Sémantique. *Doctoral dissertation*, Université de Grenoble, 2012. Available at: `http://tel.archives-ouvertes.fr/tel-00834448/`, retrieved October 2013.

(5) Hasemann, H., Kroller, A., Pagel, M. RDF Provisioning for the Internet of Things. In 3^{rd} *International Conference on the Internet of Things* (IOT), pp. 143-150, IEEE, 2012.

(6) Jagalpure, A. G. RGIS: Efficient Representation, Indexing and Querying of Large RDF Graphs. *Master thesis*, University of Georgia, Athens, 2012. Available at: `https://getd.libs.uga.edu/pdfs/jagalpure_aniruddha_g_201212_ms.pdf`, retrieved October 2013.

(7) Peroni, S., Poggi, F., Vitali, F. Tracking changes through EARMARK: a theoretical perspective and an implementation. In 1^{st} *International Workshop on Document Changes: Modeling, Detection, Storage and Visualization* (DChanges), 2013. Available at: `http://ceur-ws.org/Vol-1008/paper6.pdf`, retrieved October 2013.

(8) Brunsmann, J. Long term preservation of product lifecycle metadata in OAIS archives, *Doctoral dissertation*, FernUniversität in Hagen, 2012. Available at: `http://deposit.fernuni-hagen.de/2798/`, retrieved October 2013.

(9) Joshi, A. K., Hitzler, P., Dong, G. Towards Logical Linked Data Compression. In the *Joint Workshop on Large and Heterogeneous Data and Quantitative Formalization in the Semantic Web* (LHD+ SemQuant), 2012.

(10) Leblay, J. SPARQL query answering with bitmap indexes. In 4^{th} *International Workshop on Semantic Web Information Management*, article 9, ACM, 2012.

(11) Rousset, M. M. C. Static Analysis of Semantic Web Queries. *Doctoral dissertation*, Université de Grenoble, 2012. Available at: `http://tel.archives-ouvertes.fr/docs/00/83/44/48/PDF/these_de_Melisachew_Wudage_CHEKOL.pdf`, retrieved October 2013.

(12) Skritek, S. Foundational aspects of semantic web optimization. In *SIGMOD/PODS*, 2012 PhD Symposium, pp. 45-50, ACM, 2012.

(13) Brisaboa, N. R., Cánovas, R., Claude, F., Martínez-Prieto, M. A., Navarro, G. Compressed string dictionaries. In *Experimental Algorithms*, pp. 136-147, Springer Berlin Heidelberg, 2011.

(14) Weaver, J., Williams, G. T. Reducing I/O Load in Parallel RDF Systems via Data Compression. In 1^{st} *Workshop on High-Performance Computing for the Semantic Web* (HPCSW), Vols. CEUR-WS 736, paper 4, 2011.

National Conferences

- 2013
  - Mario Arias, Oscar Corcho, Javier D. Fernández, Miguel A. Martinez-Prieto, Mari Carmen Suárez-Figueroa. Compressing Semantic Metadata for Efficient Multimedia Retrieval. In 15th *Conference of the Spanish Association for Artificial Intelligence* (CAEPIA), to be published, 2013. Available at: `http://dataweb.infor.uva.es/wp-content/uploads/2013/06/caepia2013.pdf`, retrieved October 2013.
  - Mario Arias, Carlos E. Cuesta, Javier D. Fernández, Miguel A. Martínez-Prieto. SOLID: una Arquitectura para la Gestión de Big Semantic Data en Tiempo Real. In *XVIII Jornadas de Ingeniería del Software y Bases de Datos* (JISBD), to be published (in Spanish), 2013. Available at: `http://dataweb.infor.uva.es/wp-content/uploads/2013/06/jisbd2013.pdf`, retrieved October 2013.
  - Mario Arias, Javier D. Fernández, Miguel A. Martínez-Prieto. Aplicaciones Semánticas basadas en RDF/HDT. In *XVIII Jornadas de Ingeniería del Software y Bases de Datos* (JISBD), to be published (in Spanish), 2013. Available at: `http://dataweb.infor.uva.es/wp-content/uploads/2013/06/jisbd20131.pdf`, retrieved October 2013.
- 2011
  - Javier D. Fernández, Miguel A. Martínez-Prieto, Mario Arias, Claudio Gutierrez, Sandra Álvarez-García, Nieves R. Brisaboa. Lightweighting the Web of Data through Compact RDF/HDT. In 1st *Workshop en Tecnologías de Linked Data y sus aplicaciones en España* (TLDE 2011), 2011. 14th *Conference of the Spanish Association for Artificial Intelligence*, (CAEPIA), pp. 483-493, LNCS 7023, Springer-Verlag, 2011.

    This paper has been cited by:

    (1) Kunze, S. R., Auer, S. Dataset Retrieval. In 7th *IEEE International Conference on Semantic Computing* (ICSC 2013), to be published, 2013.
  - Javier D. Fernández, Miguel A. Martínez-Prieto, Claudio Gutierrez. Compact Representation of Large RDF Data Sets for publishing and exchange. In *XVI Jornadas de Ingeniería del Software y Bases de Datos* (JISBD 2011), 2011.
  - Javier D. Fernández, Miguel A. Martínez-Prieto, Mario Arias, Claudio Gutierrez. HDT End-Points: una Arquitectura Eficiente para la Web de Datos. In *XVI Jornadas de Ingeniería del Software y Bases de Datos* (JISBD 2011), 2011.

Research Distinctions

- Best Management/Policy track Paper Award. 12th Annual International Conference on Digital Government Research (dg.o 2011). Publishing Open Statistical Data: the Spanish Census. In conjunction with Miguel A. Martínez-Prieto and Claudio Gutiérrez.

- Best Challenge Paper Award. 1st International Workshop on Usage Analysis and the Web of Data (USEWOD 2011). An Empirical Study of Real-World SPARQL Queries. In conjunction with Mario Arias, Miguel A. Martínez-Prieto and Pablo de la Fuente.

RESEARCH STAYS

- **August 2013-September 2013.** Department of Computer Science, Univ. of Chile (Santiago, Chile). Claudio Gutierrez, supervisor.
- **September 2012-October 2012.** Department of Computer Science, Università della Sapienza (Rome, Italy). Stefano Leonardi, supervisor.
- **March 2011-July 2011.** Department of Computer Science, Univ. of Chile (Santiago, Chile). Claudio Gutierrez, supervisor.
- **July-October 2010.** Department of Computer Science, Univ. of Chile (Santiago, Chile). Claudio Gutierrez, supervisor.
- **July 2010-January 2011.** Department of Computer Science, Univ. of Chile (Santiago, Chile). Claudio Gutierrez, supervisor.

I hate waiting. I could give you my word as a Spaniard.

The Princess Bride (1987)

B Summary (in Spanish)

B.1 Hipótesis y Objetivos

El actual estado del arte confirma la necesidad de disponer de una representación binaria de RDF, con el objetivo de reducir los altos niveles de verbosidad/redundancia y las bajas capacidades operativas actuales de los conjuntos de datos. A *nivel físico*, dicho formato binario de representación debe facilitar que el procesamiento, el manejo y el intercambio de información (tanto entre sistemas como intercambio memoria-disco) sean eficientes a gran escala. Por ello, dicho formato debe minimizar la redundancia al tiempo que garantice la *modularidad* de la representación. A *nivel operacional*, las características esperadas incluyen un soporte nativo para verificar la simple existencia de sentencias (*lookups*) así como la resolución de otros patrones simples de consulta.

Nuestra hipótesis, por tanto, puede resumirse en:

Dado un conjunto de datos RDF, potencialmente grande, un formato de RDF, binario y ligero, puede codificar los datos aprovechando la estructura sesgada de los grafos RDF, con el objetivo de conseguir (i) un notable ahorro de espacio, (ii) una publicación centrada en los datos, fácil y modular, así como (iii) soportar operaciones para la recuperación de datos.

Con esta hipótesis, proclamamos la necesidad de avanzar hacia sintaxis de RDF centradas en los datos. Proponemos un formato binario de serialización, `HDT`, que organiza la información y usa la estructura sesgada de los grafos RDF (Ding & Finin, 2006; Oren et al., 2008) para conseguir notables ahorros de espacio. Presentamos, a continuación, los principales requisitos de un formato de serialización RDF, que serán por tanto los objetivos de nuestra propuesta:

- **Deber ser generado eficientemente desde otro formato RDF y ser igualmente sencilla su conversión a otras representaciones**. Por ejemplo, un publicador de datos que mantiene la información en un almacén de datos semánticos debe ser capaz de realizar un volcado eficiente a un formato de intercambio optimizado para tal operación. De igual modo, el proceso de conversión a otro formato (potencialmente binario) puede completarse de manera más eficaz si el formato de serialización permite un recorrido de los datos eficiente.

- **Debe basarse en un esquema de publicación claro**. El formato debe mantener un esquema estándar que incluya metadatos acerca de la publicación y su contenido, junto con información relevante para recuperar el conjunto de datos.

- **Debe ser eficiente en términos de espacio**. El formato de intercambio debería generar tamaños tan reducidos como fuera posible, introduciendo para ello nociones de compresión de datos. El hecho de reducir tamaño no sólo minimiza los costes de ancho de banda para el servidor, sino que también ahorra tiempo de espera para el consumidor que desea recuperar el conjunto de datos para cualquier tipo de consumo.

- **Debe estar preparado para su posterior procesamiento**. Un caso de uso típico en casi cualquier tarea de procesamiento consiste en ejecutar una serie de lecturas secuenciales sentencia a sentencia. Aunque pueda parecer trivial, esta lectura, claramente, consume una ingente cantidad de tiempo cuando procesemos grandes volúmenes de datos en el consumidor.
- **Debería ser capaz de localizar ciertos datos concretos dentro del conjunto de datos completo**. Ante tales volúmenes de datos, sería ciertamente deseable poder evitar realizar una lectura completa de todo el conjunto de datos para localizar únicamente un dato concreto. Para ello, el formato de serialización debe contener las claves necesarias para permitir la localización de datos concretos. En particular, un formato de serialización debería ser capaz de resolver la mayoría de las combinaciones de patrones de sentencias SPARQL (combinaciones posibles de constantes o variables en sujetos, predicados u objetos). Por ejemplo, un patrón típico es proveer, únicamente, un sujeto concreto, estableciendo como resultado esperado las variables predicado y objeto. En este caso, se pretende localizar todas las sentencias que hablan de un sujeto específico[1]. En otras palabras, este requisito contiene una intención subyacente; los datos deben codificarse de tal manera que "los datos sean el índice".

B.2 Metodología

Para conseguir los objetivos perseguidos, se ha llevado a cabo una metodología de investigación en cuatro etapas cíclicas. Se ha realizado una iteración completa por año. A continuación resumimos los pasos dados en cada una de ellas a lo largo de las distintas fases.

1. **Estudio del contexto y las soluciones existentes**. Se estudió los formatos RDF existentes, las posibilidades de indexación y consulta. Para ello, se estudian en profundidad los procesos existentes en la Web de Datos, identificando un flujo de datos común de Publicación-Intercambio-Consumo, y se centra la investigación en abordar esta problemática.
2. **Detección de problemas**. En esta etapa se detectó que el rendimiento del flujo de datos anterior se encuentra muy influenciado por i) el formato de intercambio de datos y ii) los índices existentes de RDF. En primer lugar, se detectó que los formatos existentes de RDF sobrecargan de verbosidad a la representación, siendo inmanejables para grandes volúmenes de datos. Del mismo modo, la indexación y consulta de RDF se basan en estructuras auxiliares que no aprovechan todas las características de la esencia de RDF como grafo etiquetado, y su distribución sesgada.
3. Propuesta de solución. En primer lugar, se estudia la compresibilidad de RDF, proponiendo una separación en Diccionario y Triples y una compresión específica para cada componente. Se demuestra que mejora sustancialmente a los compresores existentes. En segundo lugar, se propone una estructura de representación `HDT` óptima para su publicación e intercambio eficientes. Dicha representación no sólo permite mejorar la compresibilidad (por tanto optimizando el intercambio) sino que proporciona las herramientas adecuadas para la consulta básica de RDF sin grandes estructuras auxiliares. Por ello, a continuación se propone usar `HDT` para el consumo de datos, aplicando técnicas de estructuras de datos compactas que permiten consultar los datos sin necesidad de descompresión. Se proponen dos índices complementarios creados en el consumidor de datos para poder realizar todas las operaciones básicas de consulta requeridas en SPARQL, el lenguaje estándar para consumir RDF.
4. Desarrollar la nueva solución. En esta etapa se implementa la propuesta existente. En concreto se formaliza el estándar `HDT`, y se implementan sus componentes en un prototipo de herramienta

[1]Nótese que esta consulta puede emplearse para *dereferenciar* una entidad siguiendo el tercero de los principios de Linked Data.

HDT para dar cabida a la creación de índices en el consumidor. Así, se implementa un componente de diccionario comprimido y dos índices complementarios para los triples: el primero de los índices se implementa sobre una propuesta de árbol balanceado mientras que el segundo es una lista ordenada compacta de referencias objeto.

5. Evaluar la nueva solución. Finalmente, se muestra como la solución mejora las propuestas existentes, y se evalúan las propuestas con artículos presentados ante la comunidad científica nacional e internacional.

B.3 Principales Resultados del Trabajo

La principal contribución de esta tesis es un formato novedoso para representar RDF de manera binaria, denominado HDT: *Header-Dictionary-Triples*, que aborda la publicación, intercambio y consumo (indexación/consulta) de RDF a gran escala. HDT representa la información de un conjunto de datos RDF mediante tres componentes optimizados:

- La cabecera (*Header*), incluye todo tipo de metadatos para describir el (potencialmente grande) conjunto de datos semánticos.
- El diccionario (*Dictionary*), organiza todos los identificadores (IDs) en el grafo RDF. Provee un catálogo de las entidades de información en el grafo RDF, con altos niveles de compresión.
- La estructura de sentencias RDF (*Triples*), comprende la estructura pura del grafo RDF subyacente, mitigando el ruido producido por los términos textuales, en su mayoría de gran longitud y ampliamente repetidos.

Junto con varios artículos y publicaciones científicas importantes que se detallan en el anexo de publicaciones de la tesis (ver Anexo A), cabe mencionar especialmente que la propuesta de formato estándar HDT fue presentada al Consorcio de la Web (W3C) en calidad de "Member Submission". Esta propuesta fue apoyada por ocho socios internacionales, siendo aceptada en Mayo de 2011 (Fernández et al., 2011), lo que representó un hito por el reconocimiento global de la comunidad científica y técnica en la Web Semántica.

Otras contribuciones específicas de la tesis pueden resumirse en:

1. *Marco teórico de la estructura de RDF*. En primer lugar, abordamos la problemática de comprender la estructura real de los grandes grafos RDF. Para ello, llevamos a cabo un estudio detallado de estos grafos, revelando su estructura y composición subyacentes. El principal objetivo no es otro que poder aislar características comunes que nos permitan caracterizar de manera objetiva los datos RDF del mundo real. Esta caracterización puede ser de utilidad a la hora de realizar mejores diseños de conjuntos de datos, así como en el desarrollo de estructuras de datos, índices y compresores de RDF más eficientes.

   Con este objetivo en mente, proponemos parámetros específicos para caracterizar los datos RDF. Nos centramos, específicamente, en aflorar la redundancia de cada conjunto, así como sus posibilidades de compresión. Dichos parámetros han sido evaluados en conjuntos de datos reales.

2. *Especificación de RDF binario*. Basándonos en nuestro análisis previo de los principales problemas de escalabilidad en el manejo de grandes volúmenes de datos semánticos, diseñamos, analizamos, desarrollamos y evaluamos el mencionado formato binario de RDF, denominado HDT, que da respuesta a nuestra hipótesis.

3. *Diccionarios RDF comprimidos y funcionales*. Sobre la base del diccionario HDT definido previamente, proponemos técnicas específicas para diccionarios RDF. En particular, abordamos el diseño de diccionarios RDF altamente comprimidos y que, al mismo tiempo, nos proporcionen una resolución de consultas eficiente. Para ello, adaptamos técnicas existentes en el campo de diccionarios comprimidos de cadenas. La solución propuesta, un nuevo diccionario RDF denominado $\mathcal{D}_{comp}$, se demuestra sobresaliente en espacio (consigue las mejores tasas de compresión en nuestra evaluación) y en rendimiento (frente a diccionarios tradicionales del estado del arte). Además, su rango de funcionamiento, en término de espacio/tiempo puede ser ajustado de acuerdo a las necesidades particulares, gracias a la organización en subdiccionarios que realiza $\mathcal{D}_{comp}$. Finalmente, se propone una funcionalidad más avanzada para ayudar a resolver filtros SPARQL desde el propio diccionario.

4. *Índices compactos de la estructura de sentencias RDF*. Abordamos la creación y uso de índices compactos de la estructura del grafo codificado en HDT. Diseñamos implementaciones prácticas que emplean estructuras de datos sucintas y ciertas nociones de compresión. En primer lugar, consideramos una nueva estructura de grafo para intercambio, denominada *Bitmap Triples* (BT) que codifica el grafo como un bosque de árboles, uno por cada sujeto y sus relaciones. A continuación, proponemos índices ligeros que el propio consumidor puede construir sobre la información HDT intercambiada. La configuración final de índices de la estructura del grafo (en el consumidor) se denomina BTWO*. Describimos en detalle los algoritmos para la resolución de patrones de sentencias a través de estos índices y, aún más importante, detallamos los costes operacionales a través de las métricas propuestas previamente. Todas las configuraciones han sido estudiadas y evaluadas en escenarios reales.

5. *Implementación práctica de RDF binario*. Una vez se han asentado los objetivos sobre el diccionario y la estructura del grafo, abordamos la integración eficiente de ambos componentes. Consideramos, por tanto, que HDT se serializa con sus componentes en formato $\mathcal{D}_{comp}$, para el diccionario, y BT, para la estructura del grafo. Sobre ellos, en el consumidor, se cargan las estructuras requeridas para consultar $\mathcal{D}_{comp}$, y se construyen los índices BTWO*. Esta propuesta se implementa y evalúa frente a otras soluciones existentes en el área de los almacenes de RDF. Nuestros experimentos muestran como HDT sobresale en casi todos los pasos del flujo publicación-intercambio-consumo, manteniéndose competitivo en la resolución de consultas.

References

Abadi, D. J., Adam, M., Madden, S. R., & Hollenbach, K. (2007). Scalable Semantic Web Data Management Using Vertical Partitioning. In *Proc. of the Very Large Data Bases (VLDB) Conference* (pp. 411–422).

Abadi, D. J., Madden, S. R., & Ferreira, M. (2006). Integrating Compression and Execution in Column-Oriented Database Systems. In *Proc. of the ACM SIGMOD International Conference on Management of data* (p. 671-682).

Abadi, D. J., Marcus, A., Madden, S. R., & Hollenbach, K. (2009). SW-store: a vertically partitioned DBMS for semantic Web data management. *The VLDB Journal*, *18*, 385–406.

Abramson, N. (1963). *Information theory and coding*. McGraw-Hill.

Adamic, L. A. (1999). The Small World Web. In *Proc. of the European Conference on Digital Libraries (ECDL)* (pp. 443–452).

Adida, B., Herman, I., Sporny, M., & Birbeck, M. (Eds.). (2012). *RDFa 1.1 Primer*. W3C Working Group Note. (`http://www.w3.org/TR/xhtml-rdfa-primer/`, retrieved October 2013)

Akar, Z., Halaç, T. G., Ekinci, E. E., & Dikenelli, O. (2012). Querying the Web of Interlinked Datasets using VOID Descriptions. In *Proc. of the Linked Data on the Web Workshop (LDOW)*.

Albert, R., Jeong, H., & Barabasi, A. L. (1999). Diameter of the World Wide Web. *Nature*, *401*(February), 130–131.

Alexander, K. (2008). RDF in JSON: A Specification for serialising RDF in JSON. In *Proc. of the Workshop on Scripting for the Semantic Web (SFSW)* (pp. 76–81).

Alexander, K., Cyganiak, R., Hausenblas, M., & Zhao, J. (2011). *Describing Linked Datasets with the VoID Vocabulary*. W3C Interest Group Note. (`http://www.w3.org/TR/void/`, retrieved October 2013)

Alexander, K., Cyganiak, R., Zhao, M., & Hausenblas, M. (2009). Describing Linked Datasets - On the Design and Usage of voiD, the "Vocabulary of Interlinked Datasets". In *Proc. of the Linked Data on the Web Workshop (LDOW)*.

Álvarez-García, S., Brisaboa, N. R., Fernández, J. D., & Martínez-Prieto, M. A. (2011). Compressed k2-Triples for Full-In-Memory RDF Engines. In *Proc. AMCIS*. (`http://arxiv.org/abs/1105.4004`, retrieved October 2013)

Álvarez-García, S., Brisaboa, N. R., Fernández, J. D., Martínez-Prieto, M. A., & Navarro, G. (2013). Compressed Vertical Partitioning for Full-In-Memory RDF Management. *ArXiv e-prints*. (`http://arxiv.org/abs/1310.4954`, retrieved October 2013)

Anderson, C., & Andersson, M. P. (2007). *Long tail*. Bonnier fakta.

Anglés, R., & Gutiérrez, C. (2005). Querying RDF Data from a Graph Database Perspective. In *Proc. 2nd European Semantic Web Conference (ESWC)* (pp. 346–360).

Anglés, R., & Gutiérrez, C. (2008). The Expressive Power of SPARQL. In *Proc. of the International Semantic Web Conference (ISWC)* (pp. 114–129).

Arenas, M., Bertails, A., Prud'hommeaux, E., & Sequeda, J. (Eds.). (2012). *A Direct Mapping of Relational Data to RDF*. (`http://www.w3.org/TR/rdb-direct-mapping/`, retrieved October 2013)

Arias, M., Corcho, O., Fernández, J. D., Martínez-Prieto, M. A., & Suárez-Figueroa, M. C. (2013). Compressing Semantic Metadata for Efficient Multimedia Retrieval. In *Proc. of CAEPIA*. (To appear.)

Arias, M., Fernández, J. D., Martínez-Prieto, M. A., & de la Fuente, P. (2011). An Empirical Study of Real-World SPARQL Queries. In *Proc. of USEWOD*. (`http://arxiv.org/abs/1103.5043`, retrieved October 2013)

Arz, J., & Fischer, J. (2013). LZ-Compressed String Dictionaries. *CoRR*, *abs/1305.0674*. (`http://arxiv.org/abs/1305.0674`, retrieved October 2013)

Atemezing, G., Corcho, O., Garijo, D., Mora, J., Poveda-Villalón, M., Rozas, P., . . . Villazón-Terrazas, B. (2012). Transforming Meteorological Data into Linked Data. *Semantic Web Journal*. (To appear.

`http://www.semantic-web-journal.net/system/files/swj281_1.pdf`, retrieved October 2013)

Atre, M., Chaoji, V., Zaki, M. J., & Hendler, J. A. (2010). Matrix "Bit" loaded: a scalable lightweight join query processor for RDF data. In *Proc. of the World Wide Web Conference (WWW)* (pp. 41–50).

Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., & Ives, Z. (2007). DBpedia: A Nucleus for a Web of Open Data. In *Proc. of the 6th International Semantic Web Conference (ISWC)* (pp. 11–15).

Bachlechner, D., & Strang, T. (2007). Is the Semantic Web a Small World? In *Proc. of the International Conference on Internet Technologies and Applications (ITA)* (pp. 413–422).

Baeza-Yates, R., Hurtado, C., & Mendoza, M. (2007, October). Improving search engines by query clustering. *J. Am. Soc. Inf. Sci. Technol.*, *58*(12), 1793–1804.

Baeza-Yates, R., & Ribeiro-Neto, B. (2011). *Modern Information Retrieval - the concepts and technology behind search* (2nd ed.). Pearson Education Ltd.

Bayer, R., & McCreight, E. (1970). Organization and maintenance of large ordered indices. In *Proc. of ACM SIGFIDET* (pp. 107–141).

Beckett, D. (2004). *RDF/XML Syntax Specification (Revised).* (`http://www.w3.org/TR/REC-rdf-syntax/`, retrieved October 2013)

Beckett, D., & Berners-Lee, T. (2011). *Turtle - Terse RDF Triple Language.* (`http://www.w3.org/TeamSubmission/turtle/`, retrieved October 2013)

Bell, T. C., Cleary, J. G., & Witten, I. H. (1990). *Text compression.* Prentice Hall.

Bender, M., Farach-Colton, M., & Kuszmaul, B. (2006). Cache-oblivious string B-trees. In *Proc. of PODS* (pp. 233–242).

Berners-Lee, T. (1996). WWW: Past, Present, and Future. *IEEE Computer*, *29*(10), 69-77.

Berners-Lee, T. (1998). *Notation3.* (`http://www.w3.org/DesignIssues/Notation3`, retrieved October 2013)

Berners-Lee, T. (2006). *Linked Data: Design Issues.* (`http://www.w3.org/DesignIssues/LinkedData.html`, retrieved October 2013)

Berners-Lee, T., Fielding, R., & Masinter, L. (2005). *RFC 3986, Uniform Resource Identifier (URI): Generic Syntax.*

Berners-Lee, T., Hendler, J. A., & Lassila, O. (2001). The Semantic Web. *Scientific American Magazine*, *284*(5), 28–37.

Binna, R., Gassler, W., Zangerle, E., Pacher, D., & Specht, G. (2011). SpiderStore: A native main memory approach for graph storage. In *Proc. 23rd Workshop Grundlagen von Datenbanken (GvDB)* (pp. 91–96).

Binnig, C., Hildenbrand, S., & Färber, F. (2009). Dictionary-based Order-preserving String Compression for Main Memory Column Stores. In *Proc. of the ACM SIGMOD International Conference on Management of data* (pp. 283–296).

Bishop, B., Kiryakov, A., Ognyanoff, D., Peikov, I., Tashev, Z., & Velkov, R. (2011). OWLIM: A family of scalable semantic repositories. *Semantic Web*, *2*(1), 33–42.

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

Bizer, C., Heath, T., Idehen, K., & Berners-Lee, T. (2008). Proc. of the Linked Data on the Web Workshop (LDOW). In *WWW* (pp. 1265–1266).

Bloomberg, J. (2013). *The Big Data Long Tail.* (`http://www.devx.com/blog/the-big-data-long-tail.html`, retrieved October 2013)

Bönström, V., Hinze, A., & Schweppe, H. (2003). Storing RDF as a Graph. In *Proc. 1st Latin American Web Congress (LA-WEB)* (pp. 27–36).

Borges, J., & Levene, M. (2000). Data Mining of User Navigation Patterns. In B. Masand & M. Spiliopoulou (Eds.), *Web Usage Analysis and User Profiling* (Vol. LNAI 1836, pp. 92–112). Springer Berlin Heidelberg.

Brickley, D. (2004). *RDF Vocabulary Description Language 1.0: RDF Schema*. (`http://www.w3.org/TR/rdf-schema/`, retrieved October 2013)

Brisaboa, N. R., Cánovas, R., Claude, F., Martínez-Prieto, M. A., & Navarro, G. (2011). Compressed String Dictionaries. In *Proc. of the Symposium on Experimental Algorithms (SEA)* (pp. 136–147).

Brisaboa, N. R., Ladra, S., & Navarro, G. (2013). DACs: Bringing Direct Access to Variable-Length Codes. *Information Processing and Management (IPM)*, *49*(1), 392–404.

Brisaboa, N. R., Ladra, S., & Navarro, G. (2014). Compact Representation of Web Graphs with Extended Functionality. *Information Systems*, *39*(1), 152–174.

Broder, A., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R., … Wiener, J. (2000). Graph structure in the Web. *Computer Networks*, *33*(1-6), 309–320.

Broekstra, J., Kampman, A., & van Harmelen, F. (2003). Spinning the Semantic Web. In (pp. 197–222). MIT Press.

Burrows, M., & Wheeler, D. (1994). *A Block-Sorting Lossless Data Compression Algorithm* (Tech. Rep. No. 124). Digital Equipment Corporation.

Campinas, S., Perry, T. E., Ceccarelli, D., Delbru, R., & Tummarello, G. (2012). Introducing RDF Graph Summary with Application to Assisted SPARQL Formulation. *Proc. of the 23rd International Workshop on Database and Expert Systems Applications (DEXA)*, 261-266.

Carothers, G. (2013). *N-Quads*. W3C Working Group Note. (`http://www.w3.org/TR/n-quads/`, retrieved October 2013)

Carroll, J. J. (2003). Signing RDF Graphs. In *ISWC 2003* (pp. 369–384).

Cheng, G., Ge, W., & Qu, Y. (2008). Falcons: searching and browsing entities on the semantic web. In *Proc. of the World Wide Web Conference (WWW)* (pp. 1101–1102).

Cheng, G., & Qu, Y. (2008). Term Dependence on the Semantic Web. In *Proc. of the International Semantic Web Conference (ISWC)* (pp. 665–680).

Chong, E., Das, S., Eadon, G., & Srinivasan, J. (2005). An efficient SQL-based RDF querying scheme. In *Proc. of VLDB* (pp. 1216–1227).

Clark, D. (1996). *Compact PAT trees*. Unpublished doctoral dissertation, University of Waterloo.

Clauset, A., Shalizi, C. R., & Newman, M. E. J. (2009). Power-Law Distributions in Empirical Data. *SIAM Review*, *51*(4), 661–703.

Cleary, J. G., & Witten, I. H. (1984). Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, *32*(4), 396–402.

Compact Data Structures Library (libcds). (2012). (`http://libcds.recoded.cl/`, retrieved October 2013)

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). *Introduction to Algorithms*. MIT Press.

Cuesta, C. E., Martínez-Prieto, M. A., & Fernández, J. D. (2013). Towards an Architecture for Managing Big Semantic Data in Real-Time. In K. Drira (Ed.), *Software Architecture* (Vol. 7957, pp. 45–53). Springer Berlin Heidelberg.

Cyganiak, R. (2005). A relational algebra for SPARQL. *Digital Media Systems Laboratory HP Laboratories Bristol. HPL-2005-170*. (`http://www.hpl.hp.com/techreports/2005/HPL-2005-170.html`, retrieved October 2013)

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

Cyganiak, R., & Reynolds, D. (Eds.). (2013). *The RDF Data Cube Vocabulary*. W3C Working Draft. (`http://www.w3.org/TR/vocab-data-cube/`, retrieved October 2013)

Cyganiak, R., Stenzhorn, H., Delbru, R., Decker, S., & Tummarello, G. (2008). Semantic Sitemaps: Efficient and Flexible Access to Datasets on the Semantic Web. In *Proc. of the Extended Semantic Web Conference (ESWC)* (pp. 690–704).

De, S., Elsaleh, T., Barnaghi, P. M., & Meissner, S. (2012). An Internet of Things Platform for Real-World and Digital Objects. *Scalable Computing: Practice and Experience*, *13*(1), 45–57.

Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *Communications of the ACM*, *51*(1), 107–113.

Ding, L., & Finin, T. (2006). Characterizing the Semantic Web on the Web. In *Proc. of the International Semantic Web Conference (ISWC)* (pp. 242–257).

Dorogovtsev, S. N., & Mendes, J. F. F. (2003). *Evolution of Networks: From Biological Nets to the Internet and WWW* (Vol. 51). Oxford University Press.

DuCharme, B. (2011). *Learning SPARQL*. O'Reilly.

Duerst, M., & Suignard, M. (2005, 1). RFC 3987, Internationalized Resource Identifiers (IRIs) [RFC]. IETF.

Dumbill, E. (2012). What is Big Data? *Strata*(January, 11). (`http://strata.oreilly.com/2012/01/what-is-big-data.html`, retrieved October 2013)

Erling, O. (2012). Virtuoso, a Hybrid RDBMS/Graph Column Store. *IEEE Data Eng. Bull.*, *35*(1), 3–8.

Erling, O., & Mikhailov, I. (2007). RDF Support in the Virtuoso DBMS. In *Proc. of the Conference on Social Semantic Web (CSSW)* (pp. 59–68).

Erling, O., & Mikhailov, I. (2009). RDF Support in the Virtuoso DBMS. In T. Pellegrini, S. Auer, K. Tochtermann, & S. Schaffert (Eds.), *Networked Knowledge - Networked Media* (Vol. 221, pp. 7–24). Springer Berlin Heidelberg.

Faloutsos, M., Faloutsos, P., & Faloutsos, C. (1999). On power-law relationships of the Internet topology. *ACM SIGCOMM Computer Communication Review*, *29*(4), 251–262.

Fernández, J. D., Arias, M., Martínez-Prieto, M. A., & Gutiérrez, C. (2013). Management of Big Semantic Data. In R. Akerkar (Ed.), *Big Data Computing* (chap. 4). Taylor and Francis/CRC.

Fernández, J. D., Gutiérrez, C., & Martínez-Prieto, M. A. (2010). RDF Compression: Basic Approaches. In *WWW* (pp. 1091–1092).

Fernández, J. D., Martínez-Prieto, M. A., & Gutiérrez, C. (2010). Compact Representation of Large RDF Data Sets for Publishing and Exchange. In *Proc. of the International Semantic Web Conference (ISWC)* (pp. 193–208).

Fernández, J. D., Martínez-Prieto, M. A., Gutiérrez, C., & Polleres, A. (2011). *Binary RDF Representation for Publication and Exchange (HDT)*. (`http://www.w3.org/Submission/2011/03/`, retrieved October 2013)

Fernández, J. D., Martínez-Prieto, M. A., Gutiérrez, C., Polleres, A., & Arias, M. (2013). Binary RDF Representation for Publication and Exchange (HDT). *Journal of Web Semantics*, *19*, 22–41.

Ferragina, P., Grossi, R., Gupta, A., Shah, R., & Vitter, J. S. (2008). On searching compressed string collections cache-obliviously. In *Proc. of the 27th symposium on principles of database systems (pods)* (pp. 181–190).

Ferragina, P., & Manzini, G. (2000). Opportunistic Data Structures with Applications. In *Proc. of the IEEE Symposium on Foundations of Computer Science (FOCS)* (pp. 390–398).

Ferragina, P., & Venturini, R. (2010). The compressed permuterm index. *ACM Trans. Alg.*, *7*(1), art. 10.

Foulonneau, M. (2011). Smart Semantic Content for the Future Internet. In *Metadata and Semantic Research* (Vol. CCIS 240, pp. 145–154). Springer Berlin Heidelberg.

García-Silva, A., Corcho, O., Alani, H., & Gómez-Pérez, A. (2012). Review of the state of the art: discovering and associating semantics to tags in folksonomies. *The Knowledge Engineering Review*, *27*(01), 57-85.

Garfield, E. (1976). The permuterm subject index: An autobiographical review. *Journal of the American Society for Information Science*, *27*(5), 288–291.

Garlik, S. H., Seaborne, A., & Prud'hommeaux, E. (2013). *SPARQL 1.1 Query Language*. W3C Recommendation. (`http://www.w3.org/TR/sparql11-query/`, retrieved October 2013)

Ge, W., Chen, J., Hu, W., & Qu, Y. (2010). Object Link Structure in the Semantic Web. In *Proc. of the Extended Semantic Web Conference (ESWC)* (pp. 257–271).

Gil, R., & García, R. (2004). Measuring the Semantic Web. *AIS SIGSEMIS Bulletin*, *1*(2), 69–72.

Gil, Y., & Groth, P. (2011). LinkedDataLens: linked data as a network of networks. In *Proc. of the International Conference on Knowledge Capture (K-CAP)* (pp. 191–192).

Golynski, A., Grossi, R., Gupta, A., Raman, R., & Rao, S. S. (2007). On the Size of Succinct Indices. In L. Arge, M. Hoffmann, & E. Welzl (Eds.), *Algorithms - ESA 2007* (Vol. 4698, pp. 371–382).

González, R., Grabowski, S., Mäkinen, V., & Navarro, G. (2005). Practical Implementation of Rank and Select Queries. In *Proc. of the Workshop on Efficient and Experimental Algorithms (WEA)* (pp. 27–38).

Goodman, E., Jimenez, E., Mizell, D., Al-Saffar, S., Adolf, B., & Haglin, D. J. (2011). High-performance computing applied to semantic databases. In *The Semantic Web: Research and Applications* (pp. 31–45). Springer.

Govindan, R., & Tangmunarunkit, H. (2000). Heuristics for Internet Map Discovery. *Proc. of the Annual IEEE International Conference on Computer Communications (INFOCOM)*, 1371–1380.

Grant, J., & Beckett, D. (2004). *RDF Test Cases*. (`http://www.w3.org/TR/rdf-testcases/`, retrieved October 2013)

Groppe, S. (2011). *Data Management and Query Processing in Semantic Web Databases*. Springer.

Grossi, R., Gupta, A., & Vitter, J. S. (2003). High-order entropy-compressed text indexes. In *Proc. of the Symposium on Discrete Algorithms (SODA)* (pp. 841–850).

Grossi, R., & Ottaviano, G. (2012). Fast Compressed Tries through Path Decompositions. In *Proc. of the 14th Meeting on Algorithm Engineering & Experiments (ALENEX)* (pp. 65–74).

Guéret, C., Groth, P., Van Harmelen, F., & Schlobach, S. (2010). Finding the Achilles Heel of the Web of Data: using network analysis for link-recommendation. In *Proc. of the International Semantic Web Conference (ISWC)* (pp. 289–304).

Guns, R. (2008). Unevenness in Network Properties on the Social Semantic Web. *Scalable Computing: Practice and Experience*, *9*(4), 271–279.

Gutiérrez, C. (2011). Reasoning Web. Semantic Technologies for the Web of Data - 7th International Summer School 2011, Galway, Ireland, August 23-27, 2011, Tutorial Lectures. In Polleres, A. and d'Amato, C. and Arenas, M. and Handschuh, S. and Kroner, P. and Ossowski, S. and Patel-Schneider, P.F. (Ed.), *Reasoning Web* (Vol. 6848, pp. 416–444). Springer.

Gutiérrez, C., Hurtado, C., Mendelzon, A. O., & Perez, J. (2011). Foundations of Semantic Web Databases. *Journal of Computer and System Sciences*, *77*, 520–541.

Haas, K., Mika, P., Tarjan, P., & Blanco, R. (2011). Enhanced results for web search. In *Proc. of the 34th International Conference on Research and Development in Information Retrieval (SIGIR)* (pp. 725–734).

Halfon, A. (2012). *Handling Big Data Variety*. (`http://www.finextra.com/community/fullblog.aspx?blogID=6129`, retrieved October 2013)

Harris, S., & Gibbins, N. (2003). 3Store: Efficient Bulk RDF Storage. In *Proc. 1st International Workshop on Practical and Scalable Semantic Systems (PSSS)* (pp. 1–15).

Harth, A., & Decker, S. (2005). Optimized Index Structures for Querying RDF from the Web. In *Proc. 3rd Latin American Web Congress (LA-WEB)* (p. 71-80).

Harth, A., Umbrich, J., Hogan, A., & Decker, S. (2007). YARS2: A Federated Repository for Querying Graph Structured Data from the Web. In K. Aberer et al. (Eds.), *The Semantic Web* (Vol. 4825, pp. 211–224). Springer Berlin Heidelberg.

Hausenblas, M., & Karnstedt, M. (2010). Understanding Linked Open Data as a Web-Scale Database. In *Proc. of the 1st International Conference on Advances in Databases* (pp. 56–61).

Hayes, J., & Gutiérrez, C. (2004). Bipartite Graphs as Intermediate Model for RDF. In *Proc. of the International Semantic Web Conference (ISWC)* (pp. 47–61). Springer Berlin Heidelberg.

Hayes, P. (2004). *RDF Semantics*. (`http://www.w3.org/TR/rdf-mt/`, retrieved October 2013)

Heath, T., & Bizer, C. (2011). *Linked Data: Evolving the Web into a Global Data Space*. Synthesis Lectures on the Semantic Web: Theory and Technology, Morgan & Claypool.

Hey, T., Tansley, S., & Tolle, K. M. (2009). Jim Gray on eScience: a transformed scientific method. In *The Fourth Paradigm.* Microsoft Research Redmond, WA.

Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P. F., & Rudolph, S. (2012). *OWL 2 Web Ontology Language Primer (Second Edition).* (`http://www.w3.org/TR/owl2-primer/`, retrieved October 2013)

Hogan, A. (2011). *Exploiting RDFS and OWL for Integrating Heterogeneous, Large-Scale, Linked Data Corpora*. Unpublished doctoral dissertation, National University of Ireland, Galway. (`http://sw.deri.org/~aidanh/docs/thesis/thesis-one-sided.pdf`, retrieved October 2013)

Hogan, A., Harth, A., Passant, A., Decker, S., & Polleres, A. (2010). Weaving the Pedantic Web. In *LDOW 2010 at WWW 2010.* Raleigh, USA.

Hogan, A., Polleres, A., Umbrich, J., & Zimmermann, A. (2010). 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).*

Hogan, A., Zimmermann, A., Umbrich, J., Polleres, A., & Decker, S. (2012). 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.

Hu, W., Chen, J., Zhang, H., & Qu, Y. (2011). How Matchable Are Four Thousand Ontologies on the Semantic Web. In *Proc. of the Extended Semantic Web Conference (ESWC)* (pp. 290–304).

Huang, J., Abadi, D., & Ren, K. (2011). Scalable SPARQL Querying of Large RDF Graphs. *Proceedings of the VLDB Endowment*, *4*(11), 1123–1134.

Huffman, D. A. (1952). A Method for the Construction of Minimum-Redundancy Codes. *Proc. of the Institute of Radio Engineers*, *40*(9), 1098–1101.

IBM. (1993). *IBM Dictionary of Computing*. McGraw-Hill.

Jacobson, G. J. (1988). *Succinct static data structures*. Unpublished doctoral dissertation, Carnegie Mellon University.

Janik, M., & Kochut, K. (2005). BRAHMS: A workbench RDF store and high performance memory system for semantic association discovery. In *Proc. 4th International Semantic Web Conference (ISWC)* (pp. 431–445).

Jeong, H., Mason, S. P., Barabasi, A. L., & Oltvai, Z. N. (2001, May). Lethality and centrality in protein networks. *Nature*, *411*(6833), 41–42.

Khatchadourian, S., & Consens, M. P. (2010). ExpLOD: Summary-Based Exploration of Interlinking and RDF Usage in the Linked Open Data Cloud. In *Proc. of the Extended Semantic Web Conference (ESWC)* (pp. 272–287).

Knoblock, C. A., Szekely, P., Ambite, J. L., Gupta, S., Goel, A., Muslea, M., . . . Mallick, P. (2012). Semi-Automatically Mapping Structured Sources into the Semantic Web. In *Proc. of the 9th Extended Semantic Web Conference (ESWC)* (pp. 375–390).

Knuth, D. E. (1973). *The Art of Computer Programming, volume 3: Sorting and Searching*. Addison Wesley.

Kuczma, M. (2008). *An introduction to the theory of functional equations and inequalities: Cauchy's equation and Jensen's inequality*. Springer.

Langegger, A., & Woss, W. (2009). RDFStats - An Extensible RDF Statistics Generator and Library. In *DEXA 2009* (pp. 79–83).

Larsson, N. J., & Moffat, J. A. (2000). Offline Dictionary-Based Compression. *Proc. of the IEEE*, *88*, 1722–1732.

Lassila, O., & Swick, R. R. (1999). *Resource Description Framework (RDF) Model and Syntax Specification.* `http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/`, retrieved October 2013.

Lee, J., Pham, M., Lee, J., Han, W., Cho, H., Yu, H., & Lee, J. H. (2010). Processing SPARQL queries with regular expressions in RDF databases. In *Proc. of DTMBIO* (pp. 23–30).

Le-Phuoc, D., Parreira, J. X., Reynolds, V., & Hauswrth, M. (2010). RDF On the Go : An RDF Storage and Query Processor for Mobile Devices. In *Proc. of the International Semantic Web Conference (ISWC) Posters&Demos*.

Lorenz, M. O. (1905). Methods of measuring the concentration of wealth. *Publications of the American Statistical Association*, *9*(70), 209–219.

Mäkinen, V., & Navarro, G. (2007). Rank and Select Revisited and Extended. *Theoretical Computer Science*, *387*(3), 332–347.

Mallea, A., Arenas, M., Hogan, A., & Polleres, A. (2011). On blank nodes. In *Proc. of the 10th International Conference on the Semantic Web (ISWC)* (pp. 421–437).

Manola, F., & Miller, E. (Eds.). (2004). *RDF Primer*. W3C Recommendation. (www.w3.org/TR/rdf-primer/)

Martínez-Prieto, M. A. (2010). *Estudio y aplicacion de nuevos metodos de compresion de texto orientada a palabras*. ProQuest Dissertations and Theses. (Doctoral dissertation in Spanish. `http://dataweb.infor.uva.es/wp-content/uploads/2012/02/Phd-Miguel.pdf`, retrieved October 2013)

Martínez-Prieto, M. A., Arias, M., & Fernández, J. D. (2012). Exchange and Consumption of Huge RDF Data. In *Proc. of ESWC* (pp. 437–452).

Martínez-Prieto, M. A., Fernández, J. D., & Cánovas, R. (2012a). Compression of RDF Dictionaries. In *Proc. of the ACM International Symposium on Applied Computing (SAC)* (pp. 1841–1848).

Martínez-Prieto, M. A., Fernández, J. D., & Cánovas, R. (2012b). Querying RDF Dictionaries in Compressed Space. *ACM SIGAPP Applied Computing Review*, *12*(2), 64-77.

McGuinness, D. L., & Van Harmelen, F. (2004). *OWL Web Ontology Language Overview*. W3C Recommendation. (`http://www.w3.org/TR/owl-features/`, retrieved October 2013)

Mendelzon, A. O., & Milo, T. (1998). Formal Models of Web Queries. *Inf. Syst.*, *23*(8), 615-637.

Milgram, S. (1967). The small world problem. *Psychology Today*, *2*(1), 60–67.

Miller, E. (1998). An Introduction to the Resource Description Framework. *Bulletin of the American Society for Information Science and Technology*, *25*(1), 15–19.

Motik, B., Patel-Schneider, P. F., & Parsia, B. (2009). *OWL 2 Web Ontology Language Structural Spcification and Functional Style-Syntax*. W3C Recommendation. (`http://www.w3.org/TR/owl2-syntax/`, retrieved October 2013)

Munro, I. (1996). Tables. In *Foundations of Software Technology and Theoretical Computer Science* (pp. 37–42).

Musser, J., & Oreilly, T. (2007). *Web 2.0 Report: Principles and Best Practices*. O'Reilly Media, Incorporated.

Navarro, G. (2013). Wavelet Trees for All. *Journal of Discrete Algorithms*. (To appear)

Navarro, G., & Mäkinen, V. (2007). Compressed Full-Text Indexes. *ACM Computing Surveys*, *39*(1), art. 2.

Neumann, T., & Weikum, G. (2009). Scalable Join Processing on Very Large RDF Graphs. In *Proc. of the ACM SIGMOD International Conference on Management of data* (pp. 627–640).

Neumann, T., & Weikum, G. (2010). The RDF-3X Engine for Scalable Management of RDF data. *The VLDB Journal*, *19*(1), 91–113.

Oren, E., Delbru, R., Catasta, M., Cyganiak, R., Stenzhorn, H., & Tummarello, G. (2008). Sindice.com: a document-oriented lookup index for open linked data. *International Journal of Metadata Semantics and Ontologies*, *3*(1), 37–52.

Pasco, R. C. (1976). *Source coding algorithms for fast data compression*. Unpublished doctoral dissertation, Stanford University.

Perez, J., Arenas, M., & Gutiérrez, C. (2009). Semantics and Complexity of SPARQL. *ACM Transactions on Database Systems*, *34*(3), 1-45.

Powers, S. (2003). *Practical RDF*. O'Reilly & Associates, Inc.

Prud'hommeaux, E., & Seaborne, A. (2008). *SPARQL Query Language for RDF*.

(http://www.w3.org/TR/rdf-sparql-query/, retrieved October 2013)

Quesada, J. (2008). Human Similarity theories for the semantic web. In *Proceedings of the First International Workshop on Nature Inspired Reasoning for the Semantic Web* (Vols. CEUR-WS 419, paper 7).

Ramakrishnan, R., & Gehrke, J. (2000). *Database Management Systems*. Osborne/McGraw-Hill.

Raman, R., Raman, V., & Rao, S. S. (2002). Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In *Proc. of ACM-SIAM Symposium on Discrete Algorithms (SODA)* (pp. 233–242).

Redner, S. (1998). How popular is your paper? An empirical study of the citation distribution. *European Physical Journal B*, *4*(2), 131–134.

Rissanen, J. J. (1976). Generalized kraft inequality and arithmetic coding. *IBM J. Res. Dev.*, *20*(3), 198–203.

Sakr, S., & Al-Naymat, G. (2010). Relational Processing of RDF queries: a Survey. *SIGMOD Records*, *38*, 23–28.

Sakr, S., Elnikety, S., & He, Y. (2012). G-SPARQL: a hybrid engine for querying large attributed graphs. In *Proc. 21st ACM Conference on Information and Knowledge Management (CIKM)* (pp. 335–344).

Salomon, D. (2007a). *Data Compression: The Complete Reference*. Springer-Verlag London Limited.

Salomon, D. (2007b). *Variable-length Codes for Data Compression*. Springer.

Sauermann, L., & Cyganiak, R. (2008). *Cool URIs for the Semantic Web*. W3C Interest Group Note. (http://www.w3.org/TR/cooluris/, retrieved October 2013)

Schmidt, M., Hornung, T., Küchlin, N., Lausen, G., & Pinkel, C. (2008). An Experimental Comparison of RDF Data Management Approaches in a SPARQL Benchmark Scenario. In *Proc. 7th International Conference on The Semantic Web (ISWC)* (pp. 82–97).

Schmidt, M., Hornung, T., Lausen, G., & Pinkel, C. (2008). SP2Bench: A SPARQL Performance Benchmark. *CoRR*, *abs/0806.4627*.

Schmidt, M., Meier, M., & Lausen, G. (2010). Foundations of SPARQL Query Optimization. In *Proc. of the International Conference on Database Theory (ICDT)* (pp. 4–33).

Schneider, J., & Kamiya, T. (2011). *Efficient XML Interchange (EXI) Format 1.0*. W3C Recommendation. (http://www.w3.org/TR/exi/, retrieved October 2013)

Schwarte, A., Haase, P., Hose, K., Schenkel, R., & Schmidt, M. (2011). FedX: optimization techniques for federated query processing on linked data. In *Proc. of the 10th International Conference on the Semantic Web (ISWC)* (pp. 601–616).

Sidirourgos, L., Goncalves, R., Kersten, M., Nes, N., & Manegold, S. (2008). Column-store Support for RDF Data Management: not All Swans are White. *Proceedings of the VLDB Endowment*, *1*(2), 1553–1563.

Sompolski, J., Zukowski, M., & Boncz, P. (2011). Vectorization vs. compilation in query execution. In *Proc. of the Seventh International Workshop on Data Management on New Hardware* (pp. 33–40).

Stonebraker, M., Abadi, D. J., Batkin, A., Chen, X., Cherniack, M., Ferreira, M., ... others (2005). C-store: a column-oriented DBMS. In *Proc. of the Very Large Data Bases (VLDB) Conference* (pp. 553–564).

Styles, R. (2012). *RDF, Big Data and The Semantic Web.* (http://dynamicorange.com/2012/04/24/rdf-big-data-and-the-semantic-web/, retrieved October 2013)

Taheriyan, M., Knoblock, C. A., Szekely, P., & Ambite, J. L. (2012). Rapidly Integrating Services into the Linked Data Cloud. In *Proc. of the 11th International Semantic Web Conference (ISWC)* (pp. 559–574).

Theoharis, Y., Tzitzikas, Y., Kotzinos, D., & Christophides, V. (2008). On Graph Features of Semantic Web Schemas. *IEEE Transactions on Knowledge and Data Engineering*, *20*(5), 692–702.

Tran, T., Ladwig, G., & Rudolph, S. (2013). Managing Structured and Semistructured RDF Data Using

Structure Indexes. *IEEE Transactions on Knowledge and Data Engineering*, *25*(9), 2076–2089.

Tummarello, G., Cyganiak, R., Catasta, M., Danielczyk, S., Delbru, R., & Decker, S. (2010). Sig.ma: Live views on the Web of Data. *Web Semantics: Science, Services and Agents on the World Wide Web*, *8*(4), 355–364.

Urbani, J., Maassen, J., & Bal, H. (2010). Massive Semantic Web data compression with MapReduce. In *HPDC 2010* (pp. 795–802).

Volz, J., Bizer, C., Gaedke, M., & Kobilarov, G. (2009). Discovering and Maintaining Links on the Web of Data. In *Proc. of the 9th International Semantic Web Conference (ISWC)* (pp. 650–665).

Watts, D. J. (1999). Networks, Dynamics, and the Small World Phenomenon. *American Journal of Sociology*, *105*(2), 493–527.

Weaver, J., & Williams, G. T. (2011). Reducing I/O Load in Parallel RDF Systems via Data Compression. In *Proc. of the Workshop on High-Performance Computing for the Semantic Web (HPCSW 2011)* (Vols. CEUR-WS 736, paper 4).

Weiss, C., Karras, P., & Bernstein, A. (2008). Hexastore: Sextuple Indexing for Semantic Web Data Management. *Proc. of the VLDB Endowment*, *1*(1), 1008–1019.

Wilkinson, K. (2006). Jena Property Table Implementation. In *Proc. of the International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS)* (pp. 35–46).

Wilkinson, K., Sayers, C., Kuno, H., & Reynolds, D. (2003). Efficient RDF Storage and Retrieval in Jena2. In *Proc. of the International Workshop on Semantic Web and Databases (SWDB)* (pp. 7–8).

Williams, H., & Zobel, J. (1999). Compressing Integers for Fast File Access. *The Computer Journal*, *42*, 193–201.

Witten, I. H., Moffat, A., & Bell, T. C. (1999). *Managing Gigabytes : Compressing and Indexing Documents and Images*. Morgan Kaufmann.

Wood, D. (2010). *Linking Enterprise Data*. Springer. (`http://3roundstones.com/led_book/led-contents.html`, retrieved October 2013)

Zhang, H. (2008). The scale-free nature of semantic web ontology. In *Proc. of the World Wide Web Conference (WWW)* (pp. 1047–1048).

Ziv, J., & Lempel, A. (1977). A universal algorithm for sequential data compression. *Information Theory, IEEE Transactions on*, *23*(3), 337–343.

Ziv, J., & Lempel, A. (1978). Compression of individual sequences via variable-rate coding. *Information Theory, IEEE Transactions on*, *24*(5), 530–536.