Lightweighting the Web of Data through Compact RDF/HDT

Javier D. Fernández[1], Miguel A. Martínez-Prieto[1,2], Mario Arias[1], Claudio Gutierrez[2], Sandra Álvarez-García[3], and Nieves R. Brisaboa[3]

[1] Universidad de Valladolid, España
{jfergar,migumar2}@infor.uva.es, mario.arias@gmail.com
[2] Universidad de Chile, Chile
cgutierr@dcc.uchile.cl
[3] Universidade da Coruña, España
{salvarezg,brisaboa}@udc.es

Abstract. The Web of Data is producing large RDF datasets from diverse fields. The increasing size of the data being published threatens to make these datasets hardly to exchange, index and consume. This scalability problem greatly diminishes the potential of interconnected RDF graphs. The HDT format addresses these problems through a compact RDF representation, that partitions and efficiently represents three components: Header (metadata), Dictionary (strings occurring in the dataset), and Triples (graph structure). This paper revisits the format and exploits the latest findings in triples indexing for querying, exchanging and visualizing RDF information at large scale.

1 Introduction

The Web of Data comprises very large RDF[1] datasets from diverse fields such as bioinformatics, geography or social networks. The Linked Data Project[2] has been playing a crucial role promoting the use of RDF and HTTP to publish structured data on the Web and to connect it between different data sources [5]. This philosophy has lift traditional hyperlinks to a new stage, in which more than 25 billion RDF triples are being shared and increasingly linked[3]. Linked Open Data (LOD) cloud is roughly doubling every 10 months, hence the important problem when these data need to be managed.

To date, these RDF datasets tend to be published, exchanged and consumed within plain RDF formats such as RDF/XML, N3[4] or Turtle[5], which provide human-focused syntaxes disregarding large data volumes. General compressors are used over these plain formats in order to reduce the final size, but the resultant files must be decompressed and parsed in plain at the final consumer.

[1] http://www.w3.org/TR/REC-rdf-syntax/
[2] http://linkeddata.org
[3] http://www4.wiwiss.fu-berlin.de/lodcloud/
[4] http://www.w3.org/DesignIssues/Notation3
[5] http://www.w3.org/TeamSubmission/turtle/

J.A. Lozano, J.A. Gámez, and J.A. Moreno (Eds.): CAEPIA 2011, LNAI 7023, pp. 483–493, 2011.

Several RDF indexes and RDF storages explore efficient SPARQL[6] query resolution methods [15,4]. However, these approaches suffer from lack of scalability [21]. There is still a large interest in querying optimization [19], whose performance is diminished when the RDF storages manage these very large datasets.

All this is diminishing the potential of interconnected RDF graphs due to the huge space they take in and the large time required for consuming. Thus, only a small portion of the data tend to be finally exchanged, indexed and consumed.

RDF/HDT (Header-Dictionary-Triples) addresses these issues. It proposes a binary format for publishing and exchanging RDF data at large scale [10]. This paper revisits RDF/HDT and analyzes its role in the Web of Data. We provide a set of application fields which need to overcome the aforementioned scalability problems, studying RDF/HDT in such contexts. In particular, we focus in querying, exchanging and consuming RDF, *i.e.*, the consuming usage of large RDF information. To this later end, we refer to a novel tool which consumes HDT to provide large RDF data visualization.

The paper is organized as follows. Section 2 reviews the underlying problems of large RDF in the Web of Data. Section 3 presents an overview of HDT concepts and practical issues of their implementation. We revisit HDT for indexing and querying in Section 4, studying two different solutions for Triples indexing. We provide an HDT-based architecture for RDF exchanging in Section 5. Section 6 encourages HDT for RDF consumption at large scale, referring to a visualization tool as a use-case. Finally, Section 7 concludes and addresses future challenges.

2 Related Work

The RDF data model was designed as a general framework for the description and modeling of information, hence it is not attached to a particular serialization format. RDF/XML, despite its verbosity, is useful for interchanging small-scale data. Other notations, *e.g.* Turtle and N3, allow shortening some constructions, such groups of URIs or common datatypes. However, none of these proposals seems to have considered data volume as a primary goal.

Although diverse techniques provide RDF indexes, the efficient and scalable resolution of SPARQL remains an open problem. Some of them store RDF in a relational database and perform SPARQL queries through SQL, *e.g.* Virtuoso[7]. A specific technique, called vertical-partitioning, groups triples by predicate and defines a 2-column (S,O) table for each one [21]. They allow some SPARQL queries to be speeded up, but make some others difficult, *e.g.* the queries with unbounded predicates. A different strategy is followed in RDF-3X [15] and BitMat [4]; indexes are created for all ordering combinations (SPO, SOP, PSO, POS, OPS, OSP), increasing spatial requirements.

The access points of the Web of Data, built on top of RDF, are typically the SPARQL Endpoints, services which interpret the SPARQL query language. The performance of querying this infrastructure is diminished by the aforementioned factors [18]: (1) the **response time**, affected by the efficiency of the RDF indexing structure, and (2) the overall **data exchange time**, obviously influenced by the serialization format.

[6] `http://www.w3.org/TR/rdf-sparql-query/`
[7] `http://www.openlinksw.com/dataspace/dav/wiki/Main/VOSRDF`

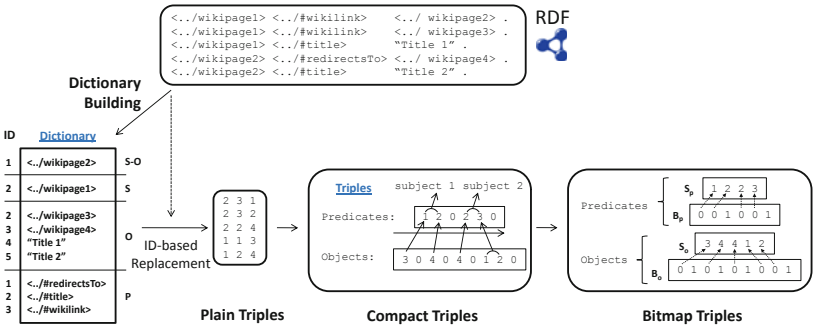

Fig. 1. Incremental representation of an RDF dataset with HDT

SPARQL resolution over the Web of Data has been addressed in two different ways. On the one hand, a federated query architecture, in the sense of traditional databases federation [20], sets up an abstraction layer of multiple SPARQL Endpoints [17]. Decomposition of queries, subquery propagation and integration of results are its main challenges. On the other hand, data centralization is based on dataset replication under a unique access point, *e.g.* the well-known Sindice service [16] or the Linked Data aggregation of OpenLink Software[8]. Both mechanisms suffer from a problem of dynamic data discovery [12] and large data management and indexing.

3 RDF/HDT

Traditional formats for serializing RDF stay influenced by the *old* document-centric perspective of the Web. This leads to fuzzy publications, inefficient management, complex processing and lack of scalability for large RDF datasets. The format RDF/HDT (*Header-Dictionary-Triples*) arises as a compact alternative to the plain formats for serializing RDF in the current Web of Data, moving forward to a data-centric scheme.

3.1 Basic Concepts

RDF/HDT[10] is a binary format for RDF recently published as a W3C Member Submission[9]. It considers the skewed structure of large RDF datasets [22] to achieve large spatial savings. It splits a dataset into three logical components:

- Header. This component is an RDF graph expressing metadata about the dataset. It extends VoiD[10] with a specific vocabulary[11] which allows logical and physical descriptions for the dataset. It can be used through well-known mechanisms, such as SPARQL Endpoints, serving as an entrance point to the information described in the dataset.

[8] http://lod.openlinksw.com/
[9] http://www.w3.org/Submission/2011/SUBM-HDT-20110330/
[10] http://www.w3.org/2001/sw/interest/void/
[11] http://www.rdfhdt.org/hdt/

Table 1. Compression results and Triples sizes for several datasets

Dataset	Triples (millions)	Size (GB)	Compression (MB)				Triples Size (MB)	
			gzip	bzip2	ppmdi	HDT-C	Bitmap	k^2-Triples
geonames	9.4	1.00	78.54	54.78	49.15	**32.36**	33.60	17.41
wikipedia	47.0	6.88	491.04	360.01	288.85	**156.40**	143.84	124.93
dbtune	58.9	9.34	924.85	630.28	441.86	**175.02**	245.78	152.58
uniprot	72.5	9.11	1233.25	739.76	637.15	**330.23**	278.59	81.92
dbpedia-en	232.5	33.12	3513.58	2645.36	2251.95	**1319.29**	995.73	884.74

- Dictionary. It maps all different strings to integer IDs. This decision pursues the goal of *compactness* because each triple can be now regarded as a group of three integer IDs.

- Triples. This component represents the graph topology by encoding all triples in the dataset. The mapping of the Dictionary allows the structure to be managed as an integer-stream. This new representation facilitates the encoder to take advantage of the existing power-law distributions for subjects and objects [10], improving HDT effectiveness.

3.2 Practical Issues

RDF/HDT supports a flexible implementation for each component depending on the final application consuming RDF. Figure 1 provides an example of different practical strategies. First, the Dictionary is built from the original RDF dataset. It is implemented on a simple hashing-based approach which distinguishes strings playing roles of: shared subject-object (`S-O`), subject (`S`), object (`O`) and predicate (`P`). Then, these mappings are used to describe three different techniques for the Triples [10].

Plain triples is the most naive approach in which only the ID replacement is carried out. **Compact triples** performs a subject ordering and creates predicate and object adjacency lists for each subject. The stream *Predicates* concatenates the predicate lists related to each subject, using the non-assigned zero ID as separator. The second stream (called *Objects*) lists all objects related to the pairs (`s,p`) in the same way. Finally, **Bitmap triples** extracts the auxiliary zero IDs embedded in each stream and stores them in two bitmaps in which 1-bits mark the end of the corresponding adjacency list.

Fernández, *et al.* [10] also proposes `HDT-Compress`, which combines specific compression techniques for the Dictionary and the Triples. Table 1 studies the effectiveness of this approach and compares them against well-known compressors for several datasets[12]. As can be seen, HDT-Compress (column HDT-C) always achieves the best compression ratio. The comparison of HDT-Compress against the typical compressor used in the area (gzip) reports improvements from 2.5 up to more than 5 times. The difference against the best compressor (ppmdi) is reduced, from 1.5 to 2.5 times, but remains significative. These results support HDT as a very compact serialization format for RDF and encourage its usage in applications, such as exchanging or publishing, in which the dataset size determines their efficiency.

[12] Geonames, dbtune and uniprot (`http://km.aifb.kit.edu/projects/btc-2010`), wikipedia (`http://labs.systemone.at`) and dbpedia (`http://wiki.dbpedia.org`)

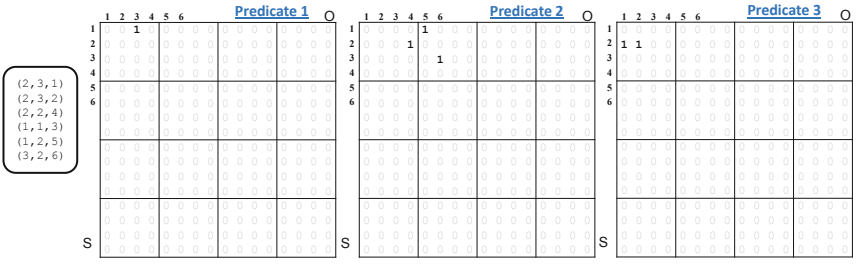

Fig. 2. k^2-triples: vertical partitioning on k^2-trees

4 RDF Indexing and SPARQL Querying

RDF indexing is a cornerstone of the Web of Data because it determines the performance of SPARQL resolution, and so, the efficiency of other tasks such as *reasoning*. A common weakness for current RDF indexing solutions is the significative time that they waste in disk transfers. Although full-in-memory indexes seem a logical solution, they are hardly scalable due to their lack of compression. In this scenario, HDT arises as an effective solution because of its compactness. This section addresses two HDT-based approaches focused on the indexing of the Triples component.

4.1 Bitmap Triples

This is an intuitive technique based on the Bitmap triples representation described in the previous section. Let us suppose a dataset containing $|S|$, $|P|$, and $|O|$ different subjects, predicates and objects respectively. Each predicate, in S_p, is represented with $log(|P|)$ bits whereas each object, in S_o, takes $log(|O|)$ bits. In turn, the bitmaps B_p and B_o are also represented in plain form [11]. This technique uses a 5% of extra space over the original bitmap length in order to achieve efficient constant time for `rank` and `select` operations [14]. These two operations enable graph structure traversing and allow some SPARQL triple pattern queries to be performed[13].

Regarding the SPO ordering, Bitmap triples resolves efficiently the triple patterns (S, P, O), $(S, P, ?O)$, $(S, ?P, ?O)$, and $(S, ?P, O)$. Note that all of them bound the subject. Patterns with unbounded subject require additional indexes to be resolved.

4.2 k^2-Triples

The *Bitmap triples* approach achieves an interesting tradeoff between compression and searching features, but it is not a full-index by itself. As we explained, the ordering chosen for its building restricts the queries than can be efficiently answered. However, its performance yields an important conclusion: compression allows RDF indexes to be fully managed in main memory, achieving very efficient SPARQL resolution.

[13] Triple patterns are like RDF triples except that each of the subject, predicate and object may be a variable. ?X values are used to indicate variable elements in the pattern.

The usage of compact data structures [14] is not very common in semantic applications. However, they have been successfully used to solve problems in areas such as Bioinformatics [7] or Web Graphs [6]. A technique from this last domain, called k^2-tree, has been generalized to be used for representing general graph databases [1]. It models a graph as a binary matrix in which a cell (`i,j`) contains `1` iff the nodes i and j are linked. This technique supports very effective resolution for queries which (1) retrieves all points in a row `x` (direct neighbours for `x`); (2) retrieves all points in a column `y` (reverse neigbours for `y`); (3) checks the existence of a given point (`x,y`); and (4) performs a bidimensional range queries involving subsets of rows and/or columns.

k^2-triples [2] uses k^2-trees to compress and index the Triples component of an RDF/HDT dataset. It is, to the best of our knowledge, the first RDF index built on compact data structures. It vertically partitions the dataset to group all triples related to a given predicate. This decision allows each group to be modeled with an independent k^2-tree which indexes all pairs (subject, object) associated with a given predicate. The resulting k^2-trees describe very sparse 1 distributions which allow k^2-triples to achieve ultra-compressed representations of the Triples component.

Figure 2 shows how k^2-triples represents the listed triples, extended from the example in Figure 1. Three independent k^2-trees are used for indexing the triples. Note that this approach works on square matrices, hence the rows/columns are expanded. In the example, only the three first rows (for the three existing subjects) and the six first columns (for the six objects) are really used in each k^2-tree, so all triples are stored in these ranges. For instance, the predicate 2 takes part in three triples: (2,2,4), (1,2,5) and (3,2,6), and its corresponding k^2-tree stores them in the coordinates (2,4), (1,5) and (3,6), which represent the corresponding subject-object pairs.

k^2-triples supports all SPARQL triple pattern queries on the primitive operations of the k^2-tree. The conjunction of these patterns allows more complex queries to be obtained through join conditions. It currently gives support for subject-subject joins, object-object and cross-joins between subjects and objects.

The results reported for k^2-triples [2] give three interesting facts: 1) it achieves the most compressed representations[14]; 2) it largely outperforms vertical partitioning on relational databases; and 3) it beats multi-index solutions for queries with bounded predicates. These results support k^2-triples for the design of full-in-memory RDF engines and its performance excels for datasets using a limited number of predicates.

5 RDF Exchanging

Communication processes in the Web of Data are threatened by the overall data exchange time. Even if current RDF formats are compressed using universal techniques, they must be decompressed at destination and then parsing the same verbose data.

HDT combines compressibility (as stated in Section 3.2) and cleaner parsing, since it already provides an index to the information. Besides establishing a compact RDF binary format for exchanging, HDT can be used as basis to design an efficient architectural model in the Web of Data. The state-of-the art reveals the need of improving the efficiency of SPARQL Endpoints, supporting (1) efficient and scalable mechanisms for

[14] Table 1 also shows that k^2-triples outperforms Bitmap triples compressibility.

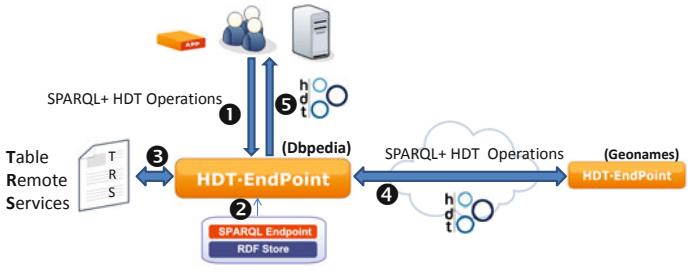

Fig. 3. Structure and communication flow in HDT·EndPoints

storing and indexing large RDF datasets, (2) compact formats for exchanging, such as HDT, and (3) protocols for discovering new resources.

HDT·EndPoints is an architecture which extends the concept of SPARQL Endpoints to support HDT functionality, taking advantage of its properties to overcome the mentioned needs. The net is built on top of HDT·Endpoint nodes.

Definition 1 (HDT·EndPoint). *An HDT·EndPoint node is an element (i) conforming to the SPARQL protocol for RDF (SPROT)*[15]*, (ii) which extends its functionality to discover and communicate with other HDT·EndPoints, and (iii) makes use of HDT as its RDF interchange format.*

Figure 3 shows the structure and communication flow for two HDT·EndPoints, storing DBpedia and Geonames in the LOD cloud. Imagine a client, (*e.g.* a human, a machine or a consuming application), who wants to retrieve all the information about "Berlin". She will send a SPARQL query to the DBpedia HDT·EndPoint (step '1' in the figure) which tries to solve it locally ('2'). Then, the DBpedia node will look up in its Table of Remote Services ('3') to discover other HDT·EndPoints which could contribute in the results. It will discover Geonames, send a subquery ('4'), harvest the results (sent in HDT) and present the final result (also in HDT) to the user ('5').

The Table of Remote Services is a mechanism to discover other HDT·EndPoints through the HDT Header. This can be seen as a "routing" table, which includes one entry per HDT dataset held in the HDT·EndPoint. Each entry stores its namespaces and the URI of the HDT·EndPoint hosting the dataset. It also includes the Header of the dataset and an optional timestamp in order to support an updating policy.

In addition to SPARQL Queries, HDT·EndPoints allows for specific HDT operations, such as returning all (of a part of) each components (Header, Dictionary, Triples).

6 Consuming RDF. Large RDF Visualization

At this point, consuming RDF is seriously underexploited [13] due to (1) the huge RDF graphs exchanging costs, (2) their complex parsing and indexing and (3) a general darkness of the underlying structure. Large RDF data tend to be complex and hard to

[15] `http://www.w3.org/TR/rdf-sparql-protocol/`

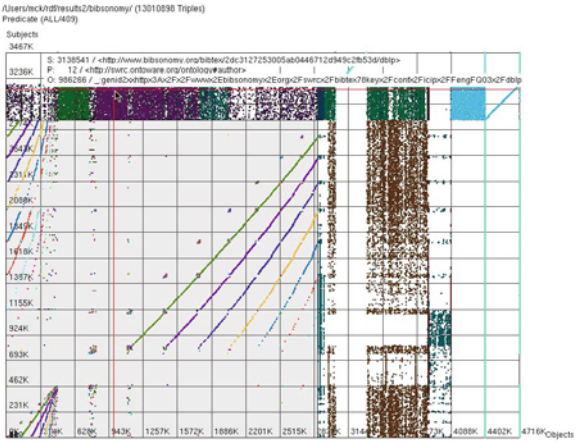

Fig. 4. Bibsonomy dataset as shown using the HDT visualization tool

read/parse in its textual publication format. Thus, semantic information developers have to deal with painful processes in order to consume these large RDF graphs.

RDF/HDT leads to compact RDF representations which not only mitigate exchanging costs, but also make the parsing and indexing easier. This way, applications consuming HDT can benefit from the reduced size as well as "instant" access to the data.

A visualization tool is proposed [3] as an example of application consuming HDT. In addition, it provides a solution for the third aforementioned problem for consumers, *i.e.* visualization and understanding of large RDF data.

6.1 Background

The use of visual tools helps consumers understand RDF content. Some of the typical tasks are identifying the most relevant resources in the graph, whether the information is grouped or scattered, or browsing the links between resources. Typical visualization tools use the node-link representation of the underlying graph. Since huge RDF datasets contain thousands to millions of statements, the number of graph nodes and edges [8] is large, causing users to have trouble interpreting the visualization. A completely different approach for rendering graph data is using its adjacency matrix [9]. It consists of generating a boolean-valued connectivity table where rows and columns represent the vertices of the graph, and each cell (x,y) states whether x is connected to y or not.

6.2 Adjacency Matrix Visualization

Arias et al. [3] proposes using a 3D adjacency matrix as an alternate visualization method for RDF. The RDF data must be available in HDT beforehand, so that the compact information can be directly consumed by the application.

The Triples component of HDT, which represents each statement as a three integer triple, can be seen as a (x, y, z) coordinate in a 3D space that can be plotted as point

in a 3D scattered plot. The y axis represents subjects, the x axis objects and the z axis predicates. The user can rotate and zoom the view to have different 3D perspectives of the data. The first and most interesting view is the one that places the camera on the z axis looking at the origin, showing a 2D figure comparing subjects against objects (Figure 4). Predicates are also highlighted using a different color for each one.

Each axis scale is annotated using the IDs, so the user can get a first sight of the amount of subjects and objects. The shared subject-object area of the dictionary is represented using a rectangle at the origin with a different background color. This area is quite interesting, because it represents the links among RDF resources.

The user experience can be enhanced by providing some extra features for interactively browsing the data. The user can hover the mouse above the graphic, showing details of the nearest triple under the cursor. The HDT Triples component can be queried to find the nearest triple, and finally the full triple can be converted back to string using the HDT Dictionary.

7 Conclusions and Future Work

RDF/HDT is a binary serialization format for RDF which decomposes the original data into three logical components: Header, Dictionary and Triples. It exploits the skewed structure of RDF datasets to achieve large spatial savings. Besides establishing a compact RDF format, HDT also provides efficient querying and parsing. We revisit HDT and study its applications in typical scenarios within the current Web of Data. We focus on indexing, querying, exchanging and consumption of large RDF datasets.

We analyze indexing and querying of HDT information through two different approaches for the Triples component, Bitmap and k^2-triples. Bitmap triples is a compact representation suitable for scalable exchange, but it only supports basic query operations. K^2-triples emerges as an ultra-compressed full-in-memory solution supporting complex SPARQL operations. Experiments show that k^2-triples is the most effective technique among all considered solutions, and the most efficient engine for solving triple patterns with bounded predicates. For future work, a query optimizer integrated with HDT would allow more complex queries to be efficiently resolved. New dictionary implementations can be also explored for providing native searches over the data, allowing to compute SPARQL filters before the triples search.

The HDT·EndPoints architecture leads to mitigate the scalability problem of the current Web of Data by means of HDT exploitation; larger volumes can be managed with smaller delays, encouraging the distribution in the Web. Furthermore, the exchange of HDT, compact and searchable, allows for direct access to the information. For future, the analyzed features of RDF/HDT open a world of possibilities and applications in the Web of Data. In particular, we envision the use of this infrastructure in mobile devices. HDT would serve, not only as the RDF transmission format, but also as an internal storage and native indexing, due to its reduced size fits mobile devices constrains.

Regarding RDF consumption, the major strength of HDT is to deal with huge datasets, achieving efficient parsing and processing, as it already embeds an index to the information. We show its applicability to a concrete problem of visualizing large-scale RDF data. The tool, based on a 3D RDF adjacency matrix, consumes and makes

use of HDT to alleviate the limitations of previous node-link graph visualization approaches. RDF consumers can benefit from the latest findings in RDF/HDT. The logical decomposition of the original RDF in three components allows for different researches, implementation and improvements for future work.

Acknowledgments. This work is funded by the MICINN of Spain TIN2009-14009-C02-02 (first three authors), Junta de Castilla y León and the European Social Fund (first author) and Institute for Cell Dynamics and Biotechnology (ICDB), Grant ICM P05-001-F, Mideplan, Chile (second author); Fondecyt 1090565 and 1110287 (fourth author); and MICINN (PGE and FEDER) TIN2009-14560-C03-02, TIN2010-21246-C02-01 and CDTI CEN-20091048, Xunta de Galicia (cofunded with FEDER) ref. 2010/17 (fifth and sixth authors), and MICINN BES-2010-039022 (FPI program), for the fifth author.

References

1. Álvarez, S., Brisaboa, N., Ladra, S., Pedreira, O.: A Compact Representation of Graph Databases. In: Proc. of MLG, pp. 18–25 (2010)
2. Álvarez García, S., Brisaboa, N., Fernández, J.D., Martínez-Prieto, M.A.: Compressed k2-Triples for Full-In-Memory RDF Engines. In: Proc. of AMCIS, TBP (2011)
3. Arias, M., Fernández, J.D., Martínez-Prieto, M.A.: RDF Visualization using a Three-Dimensional Adjacency Matrix. In: Proc. of SemSearch (2011), `http://km.aifb.kit.edu/ws/semsearch11/8.pdf`
4. Atre, M., Chaoji, V., Zaki, M.J., Hendler, J.A.: Matrix "Bit" loaded: a scalable lightweight join query processor for RDF data. In: Proc of WWW, pp. 41–50 (2010)
5. Bizer, C., Heath, T., Idehen, K., Berners-Lee, T.: Linked Data On the Web (LDOW 2008). In: Proc. of WWW, pp. 1265–1266 (2008)
6. Brisaboa, N.R., Ladra, S., Navarro, G.: k^2-Trees for Compact Web Graph Representation. In: Karlgren, J., Tarhio, J., Hyyrö, H. (eds.) SPIRE 2009. LNCS, vol. 5721, pp. 18–30. Springer, Heidelberg (2009)
7. Claude, F., Fariña, A., Martínez-Prieto, M.A., Navarro, G.: Compressed q-gram indexing for highly repetitive biological sequences. In: Proc. of BIBE, pp. 86–91 (2010)
8. Dokulil, J., Katreniakova, J.: RDF Visualization - Thinking Big. In: Proc. DEXA, pp. 459–463 (2009)
9. Fekete, J.: Visualizing networks using adjacency matrices: Progresses and challenges. In: Proc. of CAD/GRAPHICS 2009, pp. 636–638 (2009)
10. Fernández, J.D., Martínez-Prieto, M.A., Gutierrez, C.: Compact Representation of Large RDF Data Sets for Publishing and Exchange. In: Patel-Schneider, P.F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J.Z., Horrocks, I., Glimm, B. (eds.) ISWC 2010, Part I. LNCS, vol. 6496, pp. 193–208. Springer, Heidelberg (2010)
11. González, R., Grabowski, S., Makinen, V., Navarro, G.: Practical implementation of rank and select queries. In: Proc. of WEA, pp. 27–38 (2005)
12. Hartig, O., Bizer, C., Freytag, J.-C.: Executing SPARQL Queries over the Web of Linked Data. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 293–309. Springer, Heidelberg (2009)
13. Hogan, A., Harth, A., Passant, A., Decker, S., Polleres, A.: Weaving the Pedantic Web. In: Proc. of LDOW (2010)

14. Navarro, G., Mäkinen, V.: Compressed Full-Text Indexes. ACM Computing Surveys 39(1), article 2 (2007)
15. Neumann, T., Weikum, G.: The RDF-3X Engine for Scalable Management of RDF data. The VLDB Journal 19(1), 91–113 (2010)
16. Oren, E., Delbru, R., Catasta, M., Cyganiak, R., Stenzhorn, H., Tummarello, G.: Sindice.com: a document-oriented lookup index for open linked data. International Journal of Metadata Semantics and Ontologies 3(1), 37 (2008)
17. Quilitz, B., Leser, U.: Querying Distributed RDF Data Sources with SPARQL. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 524–538. Springer, Heidelberg (2008)
18. Schmidt, M., Hornung, T., Lausen, G., Pinkel, C.: SP^2Bench: A SPARQL Performance Benchmark. In: Proc. of ICDE, pp. 222–233 (2009)
19. Schmidt, M., Meier, M., Lausen, G.: Foundations of SPARQL Query Optimization. In: Proc. of ICDT, pp. 4–33 (2010)
20. Sheth, A.P., Larson, J.A.: Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. ACM Computing Surveys 22(3), 183–236 (1990)
21. Sidirourgos, L., Goncalves, R., Kersten, M., Nes, N., Manegold, S.: Column-store Support for RDF Data Management: not All Swans are White. Proc. of the VLDB Endowment 1(2), 1553–1563 (2008)
22. Theoharis, Y., Tzitzikas, Y., Kotzinos, D., Christophides, V.: On Graph Features of Semantic Web Schemas. IEEE Trans. on Know. and Data Engineering 20(5), 692–702 (2008)