Analyses of RDF Triples in Sample Datasets*

Jakub Stárka[1,2], Martin Svoboda[1,2], and Irena Mlýnková[1]

[1] XML and Web Engineering Research Group
Faculty of Mathematics and Physics, Charles University in Prague
Malostranské náměstí 25, 118 00 Prague 1, Czech Republic
Contact e-mail: {starka,svoboda,mlynkova}@ksi.mff.cuni.cz
[2] University of Economics, Prague, Czech Republic

Abstract. Linked Data principles supported especially by RDF triples appeared recently to enrich the Web of Documents by the Web of Data. However, each application that wants to process RDF triples has to deal with their distribution, dynamics and scaling. Thus, having understood structural and other features of such data, we may have better chances to propose these applications more efficiently. Especially when we consider issues of data storing, indexing and querying. The aim of this paper is to propose characteristics that appropriately capture and describe such features of RDF triples, and to provide experimental results over a few selected real-world RDF datasets.

1 Introduction

Linked Data [3] is not any particular standard, it is just a set of common practices and general rules using which we can contribute to the Web of Data that emerged recently to enrich the traditional Web of Documents. So, what are these rules? First of all, each real-world entity should be assigned a unique URL identifier; these identifiers should be dereferenceable by HTTP to obtain information about these entities; and, finally, these entity representations should be interlinked together to form a global Linked Data cloud.

Nevertheless, despite there are also other ways how to follow the mentioned Linked Data principles, the most promising is obviously the RDF standard [6]. It assumes data modelled as triples with three components: *subject*, *predicate* and *object*. These triples can also be viewed as graphs, where vertices correspond to subjects and objects, while labelled edges represent the triples themselves.

One of our ongoing research efforts should result into a proposal of a new querying system dealing with large amounts of distributed and dynamic RDF data – issues we previously identified as open problems of the existing approaches from the area of RDF triples storing, indexing and querying [11]. It is apparent that, having the knowledge about structural and other features of data we want to process, we are able to manage such data more efficiently.

* This work was supported by the Charles University Grant Agency grant 4105/2011, the Czech Science Foundation grant P202/10/0573 and the EU ICT FP7 project 257943 (LOD2 Project).

In fact, this idea predetermines the aim of this paper – we propose a set of characteristics of RDF triples and provide experimental results over several selected datasets. These characteristics capture features of individual triple components, triples themselves and also structural features of RDF graphs, while performed experiments attempt to outline the nature of real-world RDF data.

Outline First of all, in Section 2 we explain the motivation for this paper. Section 3 provides basic theoretical background and definitions of proposed RDF characteristics, while Section 4 presents results of performed experiments. In Section 5 we shortly discuss the related work, and, finally, Section 6 concludes.

2 Motivation

If we knew characteristics about data we want to process, we would have better chances to propose algorithms and data structures that could be more efficient with respect to our expectations. In other words, this idea justifies the aim of this paper. Having understood RDF triples we want to store, index and query, we can, hopefully, achieve better results. Moreover, we can also come across approaches that require sort of a configuration (e.g. Structure Index by Tran and Ladwig [12] or Summary Index by Harth et al. [5]). But how can we provide required parameters, if we do not know enough about data or queries?

Therefore, we have proposed several characteristics we find interesting to study. First of all, the majority of indexing approaches (e.g. Hexastore by Weiss et al. [13] or BitMat Index by Atre et al. [1]) proposes to store components of RDF triples and triples themselves separately (even using fairly different structures) in order to reduce space requirements. Knowledge of string features of these component values could support this practice.

The second group of characteristics worth of studying is related to query evaluation and, in particular, access patterns to individual triple components. In case of full-text querying, we usually do not care which particular triple component should match the queried value, but in case of structural querying like SPARQL [8], we need to have suitable indices allowing us to efficiently access particular components according to the prompted query. These indices can be built, for example, on nested lists (Hexastore [13]) or B^+-trees (RDF-3X by Neumann and Weikum [7]).

Finally, we can even attempt to study more complex characteristics based on structure of RDF graphs. When using SPARQL with queries based on graph patterns, we often need to do operations similar to traditional joining in relational databases, only with the difference that we are working with RDF triples, i.e. graph data. This joining can be supported by appropriate indices as well. Like, for example, precomputed paths (RDF-3X [7]) or stars (Structure Index [12]).

It is apparent that this paper cannot encompass all possible features of RDF data that influence possibilities of their processing. So, as we will see in the following section, we have proposed at least several of them (those we treat as the most important ones with respect to our research intent) and attempted to compute them over particular selected real-world datasets.

3 Analyses

Having described our motivation, we can move forward to the core part of this paper. First, we provide some essential definitions in order to describe basic knowledge and theoretical background we need to understand to correctly introduce characteristics of RDF triples and datasets we want to study.

3.1 Basic Definitions

RDF triples are composed from three components: a subject, a predicate and an object. Beside literal values, the main building block for components of these triples is based on URI (uniform resource identifier) references as they are expected by the RDF standard. However, we assume that these references are always automatically translated to full URIs.

Thus, we can introduce $\mathbb{U}$ as a domain of all possible URI values, i.e. identifiers of resources. Analogously, assume that $\mathbb{B}$ is a domain for blank nodes and $\mathbb{L}$ a domain for literals. We do not need to study the content of these domains, we only use them to restrain the allowed values of individual triple components.

Definition 1 (RDF Triple). *We say that $t = (s, p, o)$ is an* RDF triple *(or just a* triple*), if $s \in \mathbb{U} \cup \mathbb{B}$ is a* subject*, $p \in \mathbb{U}$ is a* predicate*, and $o \in \mathbb{U} \cup \mathbb{B} \cup \mathbb{L}$ is an* object*. We say that t is a* data triple *if $o \in \mathbb{L}$.*

All values (we call them *terms*) from domains $\mathbb{U}$, $\mathbb{B}$ and $\mathbb{L}$ are seen as ordinary strings. This allows us to get deeper insight into the internal structure of URIs, generally conforming to $SchemeName$ `:` $HierarchicalPart$ [`?` $Query$] [`#` $Fragment$] scheme (we came across and studied only URLs, thus we could make this simplification). First of all, having any term x, $length(x)$ denotes a *length* of x, i.e. number of symbols it is composed of.

Now, we describe how to split URI terms into two parts. Assume that $x \in \mathbb{U}$ and p is a position of the last `#` symbol in x. Then we define $prefix(x)$ as a substring of x before p and $suffix(x)$ as a substring after p. If there is no $Fragment$ part, then we analogously use the last occurrence of `/` symbol from the hierarchical part instead. This approach should capture the way how URI terms are usually used and designed by creators of data documents and ontologies.

Sets of RDF triples are commonly modelled as *RDF graphs*.

Definition 2 (RDF Graph). *Given a set of triples T, we define $\mathcal{G} = (V, T)$ to be an* RDF graph *(or just a* graph*) as follows:*

- *V is a set of graph* vertices*, where $V = \{ x \mid \exists t \in T, t = (s, p, o)$ such that $x = s$ or $x = o \}$, and*
- *T as a set of directed graph* edges *corresponds to the underlying set of triples.*

Although we use a term graph, RDF graphs are in fact directed multigraphs since there can be more edges between the same vertices. Next, given a vertex $v \in V$ and an edge $e = (s, p, o) \in T$, we say that e is an *ingoing edge* to v if $v = o$, and that e is an *outgoing edge* from v if $v = s$.

3.2 Proposed Characteristics

According to the discussed motivation, we are now able to propose several characteristics that may be useful to know about RDF data we want to store, query or process in a different way.

Term Features The first group of proposed characteristics is connected with features of individual terms in triples. First of all, the majority of existing approaches for indexing and storing RDF data attempts to find methods of reducing the space required to store the triples. For this aim we can exploit an idea that terms often repeat, or at least their substrings may often repeat across triples in a dataset.

In other words, we can inspect lengths of particular terms, either with respect to their type ($\mathbb{U}$, $\mathbb{B}$ and $\mathbb{L}$ domains), or altogether. Next, we can split terms according to our definition of their prefix and suffix parts, exploring one suitable way of finding shared substrings.

Triple Features Now, we focus on characteristics of triple components and their categorisation. Suppose that we have a set of triples T. Given a particular term x (regardless its type), we may be interested how many triples contain this term at a particular component (subject, predicate or object). In other words, given a suitable term x, we can define $Projection_{s=x}(T) = \{\, t \mid t \in T, t = (s, p, o) \text{ and } s = x \,\}$ as a *subject projection* (or just *S projection*) corresponding to the set of all triples in T having the given fixed subject value equal to x. Analogously, we can define $Projection_{p=x}(T)$ and $Projection_{o=x}(T)$ as *P projection* and *O projection* respectively. If we model T as a graph, S and P projections correspond to sets of outgoing and ingoing edges respectively.

Moreover, there is no problem extending this idea to projections on two components concurrently. Therefore, we can define *SP projection*, *PO projection* and *SO projection* analogously. For example, $Projection_{s=x,p=y}(T) = \{\, t \mid t \in T, t = (s, p, o), s = x \text{ and } p = y \,\}$ for two suitable terms x and y. In particular, the SP projection is directly connected with the issue of multivalue properties of RDF triples causing problems in relational databases.

Star Patterns Let $\mathcal{G} = (V, T)$ be a graph and $v \in V$ a vertex. We define a graph *star* to be a set of edges $\mathcal{S}_v = \mathcal{S}_v^{in} \cup \mathcal{S}_v^{out}$, where $\mathcal{S}_v^{in} = \{\, e \mid e \in T, e = (s, p, o) \text{ and } v = o \,\}$ is an *ingoing star* around v composed from ingoing edges to v, and, analogously, $\mathcal{S}_v^{out} = \{\, e \mid e \in T, e = (s, p, o) \text{ and } v = s \,\}$ is an *outgoing star* around v.

Next, we define $sig(\mathcal{S}_v)$ as a *signature* of star $\mathcal{S}_v$ (regardless full, ingoing or outgoing) to be a set of all predicates involved in a given star; in other words, $sig(\mathcal{S}_v) = \{\, x \mid t \in \mathcal{S}_v, t = (s, p, o) \text{ and } x = p \,\}$.

Given a graph $\mathcal{G}$, we can split its vertices V into disjoint sets according to star signatures. This means that two vertices $v_1, v_2 \in V$ belong to the same set, if $sig(\mathcal{S}_{v_1}) = sig(\mathcal{S}_{v_2})$. Since this classification is an equivalency relation over

V, we can call these sets as *star classes*. Analogously, we could introduce ingoing/outgoing star classes considering only ingoing/outgoing edges respectively.

Star classes and their sizes can describe uniformity of graph vertices, thus, we can base additional characteristics on the notion of stars. Apparently, their idea is connected (and inspired) by Tran et al. [12] and their Structure Index.

Path Patterns Let $\mathcal{G} = (V, T)$ be a graph for a set of triples T and v_S, $v_T \in V$ two vertices. We say that a sequence of edges $\mathcal{P}_{v_S,v_T} = \langle e_1, ..., e_n \rangle$ with *length* $n \in \mathbb{N}_0$ is a directed *path* from the *source vertex* v_S to the *target vertex* v_T, if the following conditions hold:

- First, let $\forall\, k \in \mathbb{N}$, $1 \leq k \leq n$: $e_k = (s^k, p^k, o^k)$ and $e_k \in T$.
- If $n > 0$, then $s^1 = v_S$ and $o^n = v_T$. If $n = 0$, then necessarily $v_S = v_T$.
- Next, $\forall\, k \in \mathbb{N}$, $1 \leq k < n$: $o^k = s^{k+1}$, i.e. edges follow each other.
- $\neg\, \exists\, j$, $k \in \mathbb{N}$, $1 \leq j < k \leq n$: $s^j = s^k$ or $o^j = o^k$ or $s^j = o^j$, in other words, vertices do not repeat.

Given a particular path $\mathcal{P}_{v_S,v_T}$, we can define its *signature* as a sequence of predicates of its edges, i.e. $sig(\mathcal{P}_{v_S,v_T}) = \langle p^1, ..., p^n \rangle$.

Directed paths can serve as another characteristic that is closely related to the process of evaluating queries based on SPARQL graphs patterns.

Features Summary The following listing provides a simplified overview of all characteristics over RDF triples we have proposed in this paper:

- Term lengths – length of $\mathbb{U}$ and $\mathbb{L}$ terms viewed as strings.
- Term prefixes – length of prefixes and suffixes of $\mathbb{U}$ terms.
- Data triples – ratio of data and other triples in datasets.
- Triple projections – cardinality of S, P, O and SP, PO, SO projections.
- Star patterns – sizes of graph, ingoing and outgoing star classes.
- Path patterns – path occurrences according to their signatures.

4 Experiments

In this section, we first describe publicly available datasets we have chosen for our experiments, then we provide their implementation basics and, finally, we present results over these datasets together with some general observations.

4.1 Datasets Selection

The selection of appropriate datasets is probably one of the most important issues of any experiments. The first option could be to download a representative sample of RDF triples from the entire Linked Data cloud. However, with respect to the planned usage of our querying framework, we have finally decided

to perform the experiments over a few selected datasets only. They are from different sources, cover different thematic areas and they contain several millions of triples. Although we cannot omit DBPedia as one of the most important Linked Data sources, we selected also other interesting ones. In particular, datasets that are listed in the following summary, including their abbreviations we will use in the further text:

- ACM (ACM publications[3]) – ACM proceedings dataset with author and publication information.
- DBCS (Czech DBPedia[4]) – information extracted from Czech Wikipedia infoboxes. This dataset contains less clean data, which is actually a common situation in sources that are automatically derived from non-structured data.
- DBEN (English DBPedia[5]) – information about persons (records like date and place of birth etc.) extracted from English and German Wikipedia, represented using the FOAF vocabulary.
- GO (Gene Ontology[6]) – one of the datasets of Bio2RDF project describing publicly available DNA sequences.
- MDB (Movie Database[7]) – database containing triples about actors, movies and their relationships.

4.2 Implementation Basics

We downloaded dumps of all the previously described dataset in one of these formats: RDF/XML[8], n-triples[9] or Notation 3[10]. Then we parsed these dumps using scripts[11] implemented in Java and Python.

After necessary data cleaning (some datasets contained syntax errors), we stored all obtained triples into MySQL database using Percona Server 5.5[12] running on Debian operating system.

Since we wanted to achieve efficient computation of the proposed characteristics, we designed the database schema so as to be based on three tables: the first table contains all URI prefixes, the second one full URI values, and, finally, the third one contains triples themselves. However, instead of URI terms we stored references to the second table and instead of literals it contains their MD5 hashed values together with original lengths. The simplified schema is shown in Figure 1.

[3] http://acm.rkbexplorer.com/models/acm-proceedings.rdf
[4] http://downloads.dbpedia.org/3.7-i18n/cs/infobox_properties_cs.nt.bz2
[5] http://downloads.dbpedia.org/3.7/en/persondata_en.nt.bz2
[6] http://s4.semanticscience.org/bio2rdf_download/rdf/genbank
[7] http://queens.db.toronto.edu/~oktie/linkedmdb/linkedmdb-latest-dump.nt
[8] http://www.w3.org/TR/rdf-syntax-grammar/
[9] http://www.w3.org/TR/rdf-testcases/
[10] http://www.w3.org/TeamSubmission/n3/
[11] http://ksi.mff.cuni.cz/~starka/ld_parsers.zip
[12] http://www.percona.com/software/percona-server/

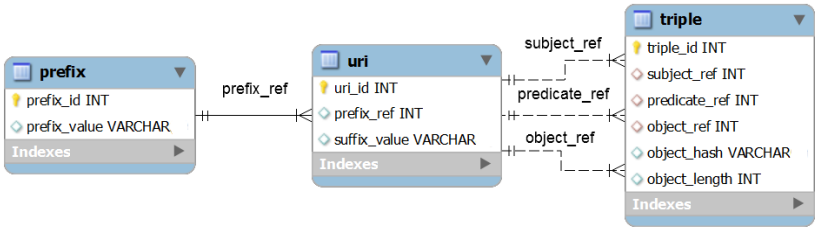


Figure 1. Database schema

4.3 Experiment Results

The majority of proposed characteristics was computed using MySQL scripts[13]. The description of the most interesting observations together with detailed experiment results is the subject of the following text.

General Characteristics Firstly, we present the basic characteristics of the data, the number of unique prefixes, URIs and triples. These results show the diversity of triples within particular datasets (see Table 1).

Table 1. Term and triple characteristics

	ACM	DBCS	DBEN	GO	MDB	Total
Term Counts						
Unique Prefixes	11	10,157	137	195	5,204	15,704
Unique URIs	810,266	162,625	867,428	1,187,775	1,327,165	4,355,259
Triple Counts						
Unique Triples	2,715,890	1,426,244	4,502,983	7,411,868	5,291,548	21,348,533
Data Triples	840,008	1,019,355	3,006,569	2,418,975	2,418,413	9,703,320
Term Lengths						
Term prefixes	31.55	54.52	48.87	57.27	47.66	52.21
Term suffixes	30.07	18.12	16.87	19.03	16.23	19.77

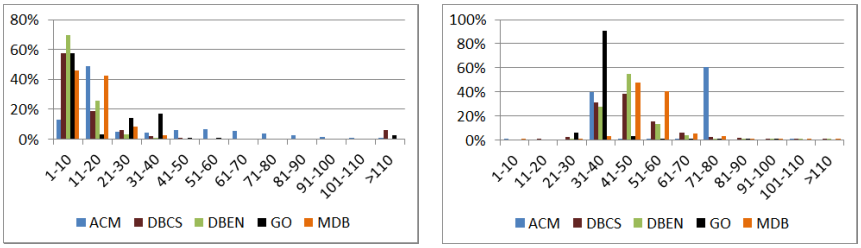
We can see that there are only 11 unique prefixes in ACM dataset, whereas there are 10,157 prefixes in DBCS dataset. These numbers suggest that ACM dataset is relatively closed (it contains mainly entities within its own domain – publications and their authors), while DBCS contains many *dirty* triples, i.e. triples where the object component is recognised as a URI but it is not a part of DBPedia (and probably neither a part of the Linked Data cloud).

The results also show the average lengths of URIs. Although there are no extreme values, we can see that ACM dataset differs. This is because the URIs (in

[13] http://ksi.mff.cuni.cz/~starka/ld_mysql.zip

all datasets) often contain artificial and often automatically generated identifiers combined with entity types and/or human readable names.

The detailed distribution of both URI and literal term lengths with respect to selected datasets can be seen in Figure 2. Since each dataset has a different total number of triples, we normalised the computed lengths by the total number of terms in each dataset.



(a) Literals (b) URIs

Figure 2. Distribution of literal and URI term lengths

As we can see, all datasets except ACM use URIs around 40 characters long. This is because identifiers in ACM dataset are padded by numeric values which causes these URIs are of the same length. On the other hand, the lengths of literals mostly range from 1 to 20 characters. This is caused by the usage of common values, i.e. person names, dates, numbers, etc. Only ACM dataset contains textual literals, in particular, keywords and concatenated lists of authors.

Triple Projections Assume that T represents a particular dataset, Θ is a comparison operator over $\mathbb{N}$ (e.g. $=$ or $>$) and $c \in \{s, p, o\}$ stands for particular triple component. Then we can define $size_{\Theta z,c} = |\{\, x : |Projection_{c=x}| \; \Theta \; z \,\}|$ as a shortcut for the number of terms x whose projections $Projection_{c=x}$ according to a particular component c have exactly z triples in case of $=$, more than z triples in case of $>$ (and analogously for the other comparison operators).

We can also define $size_{\Theta z,c_1,c_2} = |\{\, (x_1, x_2) : |Projection_{c_1=x_1,c_2=x_2}| \; \Theta \; z \,\}|$ for double projections with both $c_1, c_2 \in \{s, p, o\}$ and $c_1 \neq c_2$ as expected.

These two notions help us to present interesting features of the triple projection characteristics. In other words, we study the distribution of terms (or pairs of terms in case of double projections) according to their significance inside given datasets. For example, having the condition Θz equal to $= 1$ and inspecting the object components, we are interested in terms x such that there exists right one triple in T with x at component o. Then, $size_{=1,o}$ gives us the number of such x in T. Several projection results are presented in Table 2.

The results show, that there are usually only few unique predicates which are used in the triples. In DBCS, there are over 270 triples for each predicate, which is the lowest ratio between all datasets. In other datasets, there are thousands of triples per predicate. For subjects, the average number varies from 2 to 20.

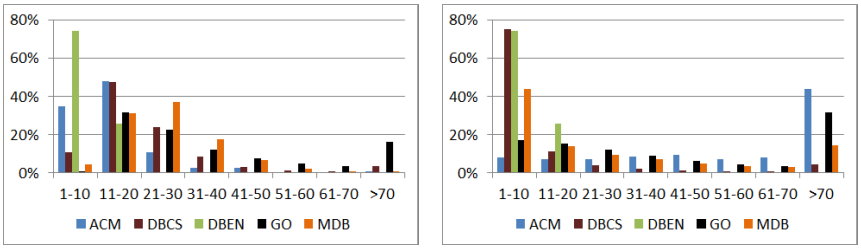
Table 2. Triple projections

	ACM	DBCS	DBEN	GO	MDB
Term Counts					
Unique subjects	810,248	68,946	790,703	776,698	694,399
Unique predicates	14	5,227	9	22	250
Unique objects	489,912	98,638	76,736	1,171,491	1,049,248
Simple Projections					
$size_{=1,o}$	403,425 (82.3%)	69,886 (70.9%)	45,551 (59.4%)	888,005 (75.8%)	657,484 (62.7%)
$size_{>1,o}$	86,487 (17.7%)	28,752 (29.1%)	31,185 (40.6%)	283,486 (24,2%)	391,764 (37.3%)
Double Projections					
$size_{=1,s,p}$	2,002,042 (89.1%)	881,317 (86.4%)	246,078 (94.2%)	6,429,816 (98.8%)	4,386,514 (94.5%)
$size_{>1,s,p}$	245,828 (10.9%)	139,254 (13.6%)	3,964,721 (5.8%)	79,925 (1.2%)	253,471 (5.5%)
$size_{=1,p,o}$	403,425 (82.3%)	102,127 (79.2%)	56,018 (61.7%)	910,420 (76.2%)	873,643 (74.1%)
$size_{>1,p,o}$	86,487 (17.7%)	26,761 (30.8%)	34,809 (38.3%)	284,764 (23.8%)	306,150 (25.9%)
$size_{=1,s,o}$	2,661,787 (99.1%)	381,569 (82.6%)	1,439,886 (64.0%)	4,980,199 (86.4%)	2,857,318 (80.9%)
$size_{>1,s,o}$	24,297 (0.9%)	80,359 (17.4%)	810,836 (36.0%)	783,039 (13.6%)	673,347 (19.1%)

In the second and third part of the table, we show projections for O and SP, PO, SO respectively. In each case we split the entire space into two disjoint parts: classes with size equal to 1 and classes with greater size. It is interesting that in most cases the projections usually have right one triple. We can also say that a typical dataset contains only a very limited number of predicates. Subjects are used mostly more than once, but they do not form large hubs.

Star Patterns Assume that T are triples of a particular dataset, then we can split vertices V of the corresponding graph $\mathcal{G} = (V, T)$ into star classes according to signatures of their star patterns, as we already know. Figure 3 depicts the distribution of star classes according to their sizes, separately for ingoing and outgoing stars. In other words, e.g. for ingoing star patterns, the horizontal axis represents different possible sizes of signatures (different numbers of predicates on ingoing edges) and the vertical axis represents the overall number of ingoing star classes having the given size.

The values are normalised in the same way as in the term length characteristic, i.e. normalised by the total number of distinct star signatures in the particular dataset.



(a) Outgoing (b) Ingoing

Figure 3. Distribution of ingoing and outgoing star class sizes

We can see that most of the unique outgoing stars have the size (i.e. number of outgoing predicates) from 10 to 30. Similarly, most of the ingoing stars have the size from 10 to 30, only except ACM dataset where sizes are distributed uniformly. Moreover, for all datasets except DBCS, the first 10% of star signatures covers more than 80% of triples.

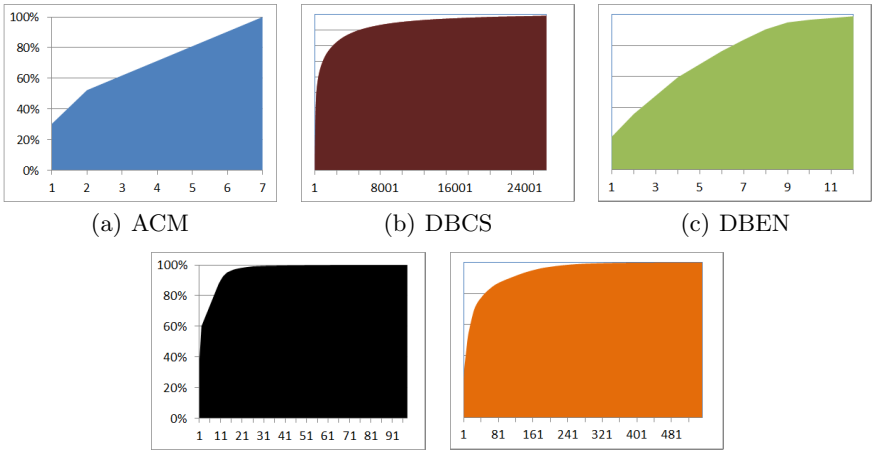
Path Patterns Similarly to star patterns, we computed also the path pattern characteristics. In particular, we considered paths of lengths equal to 2 and 3, since longer paths were out of our computation possibilities. For each path length we detected the number of unique path signatures and the overall number of all paths conforming to them, as we can see in Table 3.

Moreover, we also studied another aspect – having a particular number of the most frequent path signatures, how many paths do these signatures conform to? The number of paths with the most frequent signature is presented in the mentioned table, while the entire dependency is depicted in Figure 4.

Table 3. Statistics of path classes

Length	Property	ACM	DBCS	DBEN	GO	MDB
2	Unique signatures	7	33,394	14	55	275
	Number of paths	3,382,538	1,191,731	178	1,300,120	2,470,993
	Greatest class	1,026,874	27,786	38	247,477	248,633
3	Unique signatures	0	67,107	0	206	664
	Number of paths	0	1,428,871	0	26,863,416	15,804,941
	Greatest class	0	15,531	0	2,754,908	550,887

Finally, according to computed results, the ratio between unique path signatures and all paths themselves is relatively low. In other words, having a particular frequent signature, there are many paths conforming to it, which can be exploited in indexing techniques dealing with precomputed paths.



(a) ACM (b) DBCS (c) DBEN

(d) GO (e) MDB

Figure 4. Aggregated number of paths according to the signature frequency

5 Related Work

Although there exist several works about analyses of semantic documents and Linked Data, there are still open questions that could be discussed.

We start this overview of the related work with one of our previous papers [10], where we proposed a system for automatic document acquisition and analysis. Although we primarily focused on structural characteristics of XML documents, some basic ideas and insight into the complexity of exported datasets can be applied also in the context of Linked Data.

Ding et al. [4] described the analysis of more than 1.5 million FOAF documents. In particular, they inspected the usage of the FOAF namespace, host names and particular properties, as well as the relationships of a person in a group and other components of a social network. In general, this work describes several interesting characteristics, but its impact and context is very restrained.

Both the previous works assumed analyses at the document level, whereas Rodriguez [9] looked at datasets from the Linked Data cloud in a more complex way and computed some basic characteristics between them.

The general statistics of the Linked Data cloud are described in Bizer et al. [2]. The authors aimed at characteristics and link statistics between selected datasets. These datasets were divided by different thematic domains, for which several ingoing and outgoing statistics were computed. Provenance, licensing and dataset-level metadata published together with these datasets were also considered.

6 Conclusion

In this paper we focused on several characteristics of publicly available Linked Data datasets. The results show that although the datasets are from different areas, published by different methods and institutions, some of their characteristics are similar and, thus, the knowledge of these characteristics can be harnessed to make the management of RDF data more efficient.

We considered only a small sample of the Linked Data cloud as well as only a limited set of proposed characteristics dealing primarily with RDF triple components and structure only. On the other hand, we hope that despite this fact some observations presented in this paper can be generalised, further extended and appropriately exploited. In our future work, we plan to enrich these characteristics and also encompass a wider set of datasets and triples themselves.

References

1. 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 the 19th Int. Conf. on World Wide Web. pp. 41–50. WWW '10, ACM, NY, USA (2010)
2. Bizer, C., Jentzsch, A., Cyganiak, R.: State of the LOD Cloud (March 2011), `http://www4.wiwiss.fu-berlin.de/lodcloud/state/`
3. Bizer, C., Heath, T., Berners-Lee, T.: Linked Data – The Story so far. International Journal on Semantic Web and Information Systems 5(3), 1–22 (2009)
4. Ding, L., Zhou, L., Finin, T., Joshi, A.: How the Semantic Web is Being Used: An Analysis of FOAF Documents. In: Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05) - Track 4 - Volume 04. pp. 113–122. HICSS '05, IEEE Computer Society, Washington, DC, USA (2005)
5. Harth, A., Hose, K., Karnstedt, M., Polleres, A., Sattler, K.U., Umbrich, J.: Data Summaries for On-demand Queries over Linked Data. In: Proc. of the 19th Int. Conf. on World Wide Web. pp. 411–420. WWW '10, ACM, NY, USA (2010)
6. Manola, F., Miller, E.: RDF Primer (2004), `http://www.w3.org/TR/rdf-primer/`
7. Neumann, T., Weikum, G.: RDF-3X: A RISC-style Engine for RDF. Proc. VLDB Endow. 1, 647–659 (August 2008)
8. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF (2008), `http://www.w3.org/TR/rdf-sparql-query/`
9. Rodriguez, M.A.: A Graph Analysis of the Linked Data Cloud. CoRR (2009)
10. Starka, J., Svoboda, M., Sochna, J., Schejbal, J., Mlynkova, I., Bednarek, D.: Analyzer – A Complex System for Data Analysis. The Computer Journal (2011), *Advance Access published October 13, 2011, DOI: 10.1093/comjnl/bxr103*
11. Svoboda, M., Mlynkova, I.: Linked Data Indexing Methods: A Survey. In: On the Move to Meaningful Internet Systems: OTM 2011 Workshops. pp. 474–483. Springer (2011)
12. Tran, T., Ladwig, G.: Structure Index for RDF Data. In: Workshop on Semantic Data Management (SemData@VLDB) 2010 (2010)
13. Weiss, C., Karras, P., Bernstein, A.: Hexastore: Sextuple Indexing for Semantic Web Data Management. Proc. VLDB Endow. 1, 1008–1019 (August 2008)