Exploiting Linked Data Datasets: a Summarization Based Approach

Honghan Wu[1], Boris Villazon-Terrazas[2], Jeff Z. Pan[1], and Jose Manuel Gomez-Perez[2]

[1] Department of Computing Science, University of Aberdeen, King's College, Aberdeen, AB24 3UE, UK,
{honghan.wu,jeff.z.pan}@abdn.ac.uk
[2] iSOCO, Intelligent Software Components S.A., Av. del Partenon, 16-18, 1-7, 28042, Madrid, Spain,
bvillazon,jmgomez@isoco.com

Abstract. In the last years, we have witnessed vast increase of Linked Data datasets not only in the volume, but also in number of various domains and across different sectors. However, due to the nature and techniques used within Linked Data, it is non-trivial work for normal users to quickly understand what is within the datasets, and even for tech-users to efficiently exploit the datasets. In this paper, we propose a summarisation based approach to guide the exploitation of Linked Data. Firstly, we introduce the details of the summarisation definition and generation. Then, we present the good properties of this summarisation and propose a set of useful services from the summarisation, both of which can be utilised to guide data exploitation tasks like query writing, ontology reasoning, data compression, and data diagnosis. Finally, we evaluate our approach in several typical data exploitation scenarios. Experiments on real word datasets show that our approach can guide very efficient data exploitation.

1 Introduction

So far, Linked Data principles and practices are being adopted by an increasing number of data providers, getting as result a global data space on the Web containing hundreds of LOD datasets [1]. In this context it is important to promote the reuse and linkage of datasets, and to this end, it is necessary to know the structure of datasets. One step forward for knowing in depth the structure of a given dataset is to provide a summary of the dataset.

There are available works such as (1) *LODStats*[3] that provides the information related to a dataset, and (2) *make-void* [4] that computes statistics about RDF files. However, LODStats is thought for the whole set of LOD datasets registered in The Data Hub [5], and it is based on declarative descriptions of those datasets; and *make-void* is thought for RDF files but not for RDF datasets.

[3] http://stats.lod2.eu/
[4] https://github.com/cygri/make-void
[5] http://thedatahub.com

In this paper we present fancy-name[6], a simple tool to help users get a quick understanding of RDF dataset ...

2 Related Work

3 RDF Summarisation: The EDP Graph

Given an RDF graph, the summarization is to generate a condensed description which can facilitate data exploitations. In terms of structure, simplest graph patterns are those which only include one node. We can call these patterns as atomic patterns because it is impossible to divide them into smaller structures. By linking atomic patterns, one can construct more complex graph patterns. Following this idea, our summarization method applies a bottom-up strategy to summarize a semantic web dataset. Specifically, we propose an atomic pattern concept in which only one node is involved. Based on this concept, we summarize the given RDF dataset as a new graph which describes the relations between atomic graph patterns.

Entity Description Block A semantic web dataset is essentially an RDF graph. In such a graph, we call its non-literal nodes as entities. For such an entity e in an RDF graph G, we can get a data block for it by extracting triples in G each of which has e as its subject or object. We call such kind of data blocks as entity description block. Formally, each entity e has an entity description block (EDB for short) as defined in Definition 1.

Definition 1. *(Entity Description Block) $\forall e \in G$, the description block of e is defined as*

$$B_e = \{< e, p_i, o_i > \mid < e, p_i, o_i > \in G\} \cup \{< s_i, p_i, e > \mid < s_i, p_i, e > \in G\} \quad (1)$$

where s and p are resources in G.

Entity Description Pattern For an entity description block, its summarisation is introduced as a notion of entity description pattern (Definition 2). EDP, the short name for entity description pattern, is the atomic graph pattern in our summarisation model.

Definition 2. *(Entity Description Pattern) Given an entity description block B_e, its description pattern is a tuple $P_e = (C_e, A_e, R_e, V_e)$, where*

- *$C_e = \{c_i | < e, rdf : type, c_i > \in G\}$ is called as the class component;*
- *$A_e = \{p_i | < e, p_i, l_i > \in G$ and l_i is a literal$\}$ is called as the attribute component;*

[6] http://homepages.abdn.ac.uk/honghan.wu/pages/summ/

– *$R_e = \{r_i | < e, r_i, o_i > \in G$ and o_i is a URI resource or blank node$\}$ is called as the relation component;*
– *$V_e = \{v_i | < s_i, v_i, e > \in G\}$ is called as the reverse relation component.*

Given the EDB notion, essentially, an RDF graph G is a set of EDB i.e. $G = \cup_{e \in G} B_e$. By summarizing all entity description blocks in G, we can get the initial summarization result of G i.e. $\cup_{e \in G} P_e$. Given this initial result, we define a merge operation on EDPs which can further condense the summarization. Definition 3 defines the merge operation on EDPs which share the same class component. Based on this definition, we can merge any EDP set by grouping the EDPs in advance. Specifically, before merging, EDPs are grouped into a set of subsets according to their class components i.e. each subset contains a set of EDPs whose class components are identical and EDPs in different subsets have different class components. Then each of these subsets is merged into one EDP according to Definition 3. Finally, all merged EDPs are put together as the merging result.

Definition 3. *(EDP Merge) Given a set of EDPs:$\{P_i\}_{i=1..n}$ whose elements have identical class component C, we can merge these EDPs into a representative EDP as follows:*

$$Merge(\{P_i\}_{i=1..n}) = (C, \bigcup_{i=1..n} Attr(P_i), \bigcup_{i=1..n} Rel(P_i), \bigcup_{i=1..n} Rev(P_i)) \quad (2)$$

where

– *$Attr(P_i)$ denotes the attribute component of P_i;*
– *$Rel(P_i)$ denotes the relation component of P_i;*
– *$Rev(P_i)$ denotes the reverse relation component of P_i.*

The rationale behind this merge operation is that entities of the same type(s) might be viewed as a set of homogeneous things. This viewpoint is common in knowledge representation e.g. a class is viewed the set of all its individuals. Hence, in terms of graph pattern, the EDPs of all entities sharing the same type(s) should be merged into an integrated pattern. So far, we can define the EDP function of an RDF graph as Definition 4.

Definition 4. *(EDP of RDF Graph) Given an RDF graph G, its EDP function is defined by the following equation.*

$$EDP(G) = Merge(\bigcup_{e \in G} P_e) \quad (3)$$

EDP Graph EDP is the atomic graph pattern. Generally speaking, interesting queries usually correspond to more complex graph patterns. Hence, it would be more beneficial to know how EDPs are connected to each other in the original RDF graph. To reveal more insights about the RDF dataset in question, we introduce an EDP graph (cf. Definition 5) for characterize the linking structures in the original RDF graph.

Definition 5. *(EDP Graph) Given an RDF graph G, its EDP graph is defined as follows*

$$\begin{aligned}\mathcal{G}_{EDP}(G) =\{ &< P_i, l, P_j > | \exists e_i \in E(P_i), \exists e_j \in E(P_j), < e_i, l, e_j > \in G, \\ &P_i \in EDP(G), P_j \in EDP(G)\}\end{aligned} \tag{4}$$

where $E(P_i)$ denotes the set of entities conforms to the EDP P_i. If P_i is not merged EDP, $E(P_i)$ is the set of entities from which P_i can be generated; if P_i is a merged one, $E(P_i) = \cup_{P_k \in P} E(P_k)$, P is the set of EDPs from which P_i is merged.

As defined in Definition 5, EDP graph defines the pair-wise link relations between EDPs. From the definition, it can be figured out that if there is a link between entities of a pair of EDPs, they will have a link in the EDP graph. The EDP graph can be used to generate graph patterns or queries. Specifically, a sub-graph of the EDP graph can be converted into a query. It should be noted that if the sub-graph contains more than one links, the generated query might encounter empty result set on the original RDF graph. For example, if in the EDP graph we have $< Student, advisor, FullProfessor >$ and $< FullProfessor, headOf, Department >$, it is possible that in the RDF graph, the department head doesnt advise any student. However, it can be easily proved that EDP graph has very good properties for supporting efficient data exploitation tasks. Due to limited space, we omit the proofs. Instead different use cases will be demonstrated as use case studies.

4 The Interactive User Interface

- visualisation (The online demo: the link needs to be given)
    - Interaction Operations

5 Demos: The Summary Base Data Exploitations

- Query Generation
    - Reasoning (DL-Lite materialisation of EDP graph)
    - Data Set Enrichment

6 Conclusions and Future Work

Acknowledgments

This work was supported by K-Drive (http://www.kdrive-project.eu/).

References

1. Tom Heath and Christian Bizer. Linked data: Evolving the web into a global data space. *Synthesis Lectures on the Semantic Web Theory and Technology*, 1(1):1–136, 2011.