Schema Fragment Visualization to View Knowledge Graph Entities in Context

Sandra Siliņa[0009-0000-3917-9026], Lelde Lāce[0000-0001-7650-2355],
Mikus Grasmanis[0000-0002-0668-0970] and Kārlis Čerāns[0000-0002-0154-5294]

Institute of Mathematics and Computer Science, University of Latvia, Riga, Latvia
{sandra.silina, karlis.cerans}@lumii.lv

Abstract. We present a method for visualizing schema fragments of large knowledge graphs (KGs) to assist users in effectively exploring complex KG structures. A prototype implementation of schema fragment visualization is offered within the *ViziQuer* tool, where it complements the existing schema visualization and visual summarization methods.

Keywords: Knowledge graphs, Visual schema diagram, Schema fragments

1 Introduction

Visual presentation of a knowledge graph (KG) schema can help its users comprehend the KG structure, as well as to visually perceive the vocabulary of data classes and data properties used to encode its information.

A variety of visual KG schema presentation options exist, including visualizers of KG schema descriptions created in SHACL/ShEx (cf. [1]) or OWL (cf. [2,3]); these can be used, if the data set schema has been made available in the respective notation. There are also tools for visualizing the actual data set schema working either on-the-fly as [4], [5], or via the intermediate schema storage (cf. [6], for schema visualization inside *ViziQuer* [7]). The visualizers of SHACL data schemas can be used also in combination with the SHACL data shape retrieval from the data set (cf. [8]) to obtain a pipeline of the actual data set structure visualization, as well.

A hard limitation of the data schema visualization approach, however, is the size of the data schema that can be drawn on the diagrammatic canvas and further on perceived by a human reader. Recently, [6] has shown that heuristics of grouping together classes with similar property characteristics into a single node can effectively reduce the number of nodes to be shown within a schema diagram (e.g., 165 classes to 47 nodes for a proteomics database[1] and 267 classes to 37 nodes for Academy Sampo dataset[2], see [6] for the presentation results of 24 prominent LOD[3] data schemas). Still, even if graphs with up to 50 or in some cases 80-100 nodes can be potentially

[1] https://sparql.nextprot.org/
[2] http://ldf.fi/yoma/sparql
[3] The Linked Open Data Cloud. https://lod-cloud.net/

legibly placed on a large diagramming canvas, this might not be the best initial visual presentation of the data set to its users.

A natural consideration to construct the visual presentation of large and heterogenous datasets would be to consider their fragments. The implementation of *ViziQuer*, as described in [6], allows to specify the schema fragments either by restricting it to a number of largest data classes and properties (as used in [6] in the presentation of 5 data sets for which the visual presentation of the full schema has not been possible), or to select manually the classes and properties to be included in the diagram. Neither of these options would be convenient for the users looking for a context of a particular class resource within the data set.

In this paper we describe a method for extracting a schema fragment of a specified (class) size, centered around an initial list of data classes, from a larger data schema and presenting it to an end user, thereby giving the practical options of visually inspecting those parts of larger data schemas that are of interest. We note that a mechanistic calculation of classes linked to a given root, although possibly beneficial, can be expected to give less focused results, if compared to the analysis of the entire data schema neighborhood considered here.

A similar problem of finding concepts related to those specified by a user in an ontology has been looked at in [9], [10].

A related work considering visual presentation of schema fragments in the OWL ontology notation is [11].

The implementation described in the paper is included in *ViziQuer*[4], with playground and local setup options described on its website[5].

2 Schema Fragment Computation and Visualization

The schema fragment computation starts with selecting a single or several classes (main classes) as the starting point. In the *ViziQuer* implementation, the starting classes are manually selected within the Data Schema parameter dialogue[6], followed by specifying the fragment size (class count) and activating the 'Get Fragment' button.

This schema fragment computation approach uses a pre-computed data schema (as an enriched class-to-property relationship, as described in [6]) to find a fragment of the desired size. We consider and compare two algorithms for schema fragment computation – a heuristic one, loosely inspired by the PageRank algorithm [12], and a variant of Personalized PageRank (PPR) [12], where links of both directions are considered and weighted based on the number of class-property-class triples.

An alternative algorithm to consider is described in [9].

In the heuristic algorithm (which can be termed "Shallow PageRank"), each class has a certain relevance between 0 and 1, which influences the relevance of its neighbors.

[4] https://github.com/LUMII-Syslab/viziquer

[5] https://viziquer.lumii.lv

[6] cf. Visual Data Schema Diagrams section on https://github.com/LUMII-Syslab/viziquer/wiki

An adjacency list is used to store class-property-class links (retrieved from the data schema) for each of the classes. We refer to the classes that are selected as part of the fragment as fragment classes, and to their neighbors that are not yet selected – as candidate classes. Initially the main classes are also treated as candidate classes with infinite relevance, while all other classes have 0 relevance. This simplifies the algorithm by reducing the amount of special handling needed for the main classes while also ensuring they will be selected.

Initially, the fragment is empty and the main classes are candidates. The fragment is built by selecting classes one at a time.

In each iteration, the most relevant candidate is chosen and added to the fragment. If the chosen class is one of the main classes, its relevance is changed to 1/m (where m is the total number of main classes), ensuring proper further calculations of relevance. For each neighboring class n of the chosen class, relevance is increased by *Rc* * *cpc_c* / *cpc_cn* where *Rc* is relevance of the chosen class, and:

- *cpc_c* is the total number of class-property-class triples where one of the classes is the chosen class;
- *cpc_cn* is the number of class-property-class triples where one class is the chosen class, and the other class is its neighbor *n*.

Lastly, the set of candidates is updated to include the neighbors of the chosen class.

The fragment is complete once it has reached the desired size or there are no more available candidates.

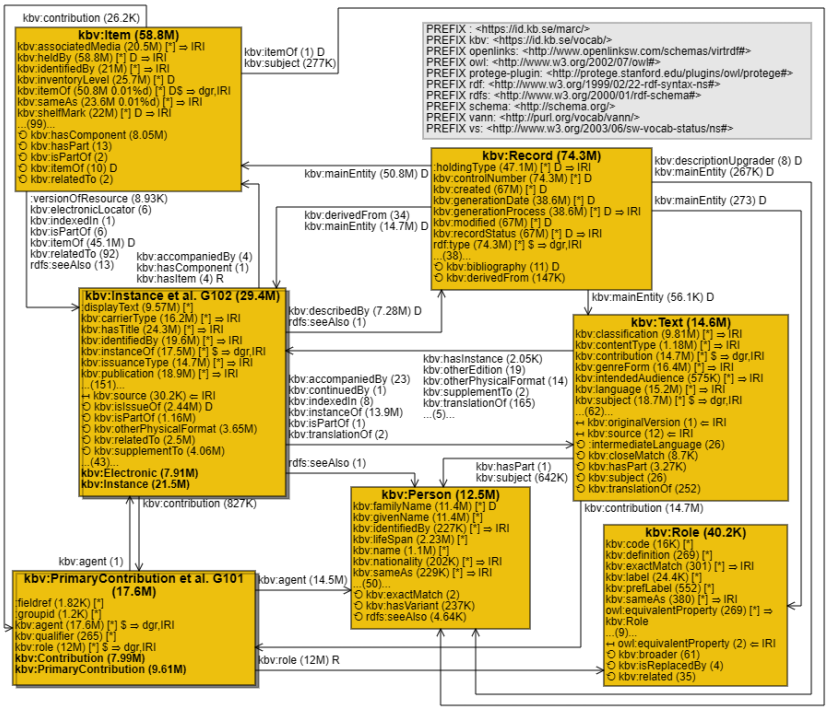

Fig. 1. A visually presented fragment of a large schema

Figure 1 shows an example of a visually presented fragment of the Swedish National Library database[7], containing, at the time of its analysis, 540 data classes, for which the means of [6] have not been available to produce a visual presentation of the entire data schema (the strongest compacting means of [6] has produced a schema graph of 329 nodes that is well above the graph visualization capacity). The fragment itself has been constructed around the class *kbv:Person*, specifying 10 as the class count. A schema diagram with 8 nodes was obtained (due to node merging); the node corresponding to *owl:Class* was deleted, resulting in the obtained 7 node diagram.

The presentation of the fragment diagram has been manually tuned after its initial automatic creation and positioning. Viewing the diagram inside the visual tool would allow its users to view full lists of class incoming, outgoing and looping attributes in the respective tool property pane widgets.

Using PPR with a damping factor of 0.85 to construct a fragment of size 10 around the class *kbv:Person* results in a similar list of classes, the only difference being that the class *kbv:Library* (with connections from *kbv:Record*, *kbv:Text*, *kbv:Item* and *kbv:PrimaryContribution*) is chosen instead of the class *kbv:Role*. This allows to assume a certain robustness of the fragment computation.

A more general comparison experiment was conducted by calculating pairs of fragments (with one fragment calculated using the Shallow PageRank algorithm and the other using PPR) surrounding each of the 100 largest classes of the Swedish National Library database. Pairs of fragments of sizes 10, 15, 20, 25 and 30 were obtained, and the percentage of shared classes was calculated for each pair. For each fragment size, cumulative values were calculated to show the number of fragment pairs that had at least a certain fraction of shared classes (see Fig. 2).

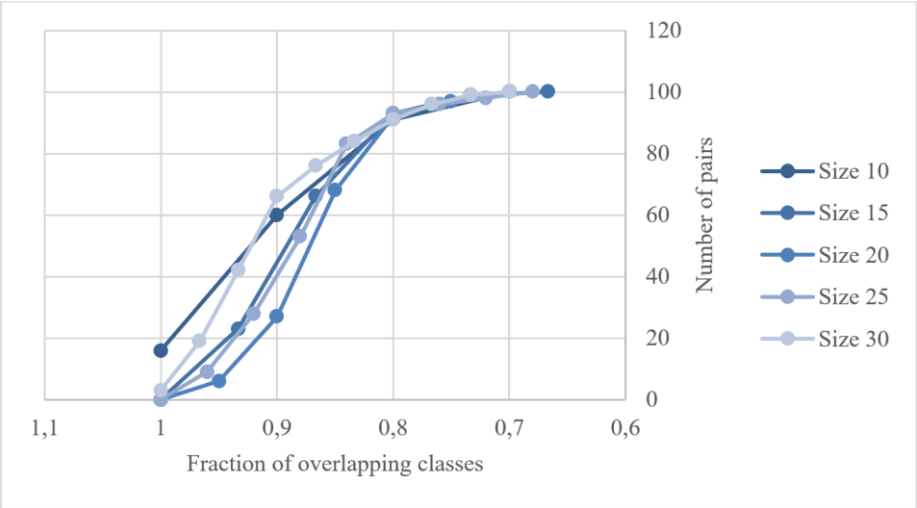

Fig. 2. Number of fragment pairs of various sizes that share at least a certain fraction of classes

[7] https://libris.kb.se/sparql

While PPR is a commonly used algorithm that may provide more accurate results due to performing many iterations to reach convergence, traditionally, the entire graph should be available. Shallow PageRank approach only requires information of those class-property-class triples where one of the classes is chosen as part of the final fragment. Therefore, with the modification of requesting information about a class's neighborhood when it is chosen to be part of the fragment instead of calculating the adjacency list initially, our approach could be a promising alternative in cases when acquiring the entire graph is too complicated or time-consuming.

For a further comparison, we also considered fragments obtained using ChatGPT[8], taking advantage of the class names in the considered dataset being human-readable and possible to interpret from the perspective of the general knowledge. The used prompt asked to, given a list of names of all available classes in the dataset, choose a certain number of classes that would likely provide the most important supplementary information about a given main class.

The resulting list of classes for *kbv:Person* differed significantly from those obtained using the two algorithms and included *kbv:Agent*, *kbv:Contribution*, *kbv:PrimaryContribution*, *kbv:Affiliation*, *kbv:Occupation*, *kbv:FieldOfActivity*, *kbv:Organization*, *kbv:Relationship* and *kbv:Identity*. We see that classes with connections to the initial class that are smaller in size (such as *kbv:Affiliation*, *kbv:Occupation* and *kbv:FieldOfActivity*, which have 437, 1.61K and 2.75K instances respectively) are also chosen. Since we are interested in the most relevant classes from the connectivity viewpoint, we consider the class lists obtained by either the Shallow PageRank algorithm or adapted PPR as suitable solutions for the schema fragment computation task.

3 Conclusions

The provided prototype implementation of schema fragment visualization has shown that obtaining and visualizing the fragments of large data schemas is a feasible task, and that it can help to create semantically focused (on the initial data classes) visual schema fragment presentations. These presentations can be expected to be of value for people doing the data analysis, including also people without deep IT experience.

Besides the UI tuning, further work on the fragment creation would involve a study on the most appropriate criteria for the class ranking for inclusion in the fragment, as well as experiments with larger data schemas, as e.g., DBPedia.

Retrieving the fragments of the schemas directly from the SPARQL endpoints (instead of using the precomputed schemas [6]) would be another powerful means in the tool chain of allowing the visual interaction of users with the KG data.

Acknowledgments

This work has been supported by the Latvian Council of Science, project "*What's in Your Knowledge Graph?*" (Project No. lzp-2024/1-0665).

[8] https://chatgpt.com/

References

1. Labra Gayo, J. E., Fernández-Álvarez, D., & Garcıa-González, H.: RDFShape: An RDF playground based on Shapes. CEUR Workshop Proceedings, vol. 2180 (2018).
2. Lohmann, S., Negru, S., Haag F., Ertl, T.: Visualizing Ontologies with VOWL. In: Semantic Web 7(4), 399-419, (2016)
3. Bārzdiņš, J., Čerāns, K., Liepiņš, R., Sproģis, A.: UML Style Graphical Notation and Editor for OWL 2. In: Proc. of BIR'2010, LNBIP, Springer 2010, vol. 64, pp. 102-113, (2010)
4. Weise, M., Lohmann, S., & Haag, F. (2016). Ld-vowl: Extracting and visualizing schema information for linked data. In Voila!2016 (pp. 120-127).
5. Dudáš, M., Svátek, V., Mynarz, J.: Dataset summary visualization with LODSight. In: The 12th Extented Semantic Web Conference (ESWC2015).
6. Lāce, L., Romāne-Ritmane, A., Grasmanis, M., Sproģis, A., Ovčiņņikova, J., Bojārs, U. and Čerāns, K. Visual Presentation and Summarization of Linked Data Schemas. In: Proc. of KGSWC 2024, Springer LNCS, Vol.15459, pp.290-305.
7. Čerāns, K., Šostaks, A., Bojārs, U., Ovčiņņikova, J., Lāce, L., Grasmanis, M., Romāne, A., Sproģis, A., Bārzdiņš, J.: ViziQuer: A Web-Based Tool for Visual Diagrammatic Queries Over RDF Data, in Springer LNCS, Vol. 11155. pp. 158–163 (2018).
8. Rabbani, K., Lissandrini, M., & Hose, K.: Extraction of validating shapes from very large knowledge graphs. In Proc. of the Very Large Databases 2023, 16(5), pp.1023-1032.
9. Chen, Y., Yang, X., Yin, K.: Identifying potentially user-interested concepts in an ontology. In: 2009 Asia-Pacific Conference on Computational Intelligence and Industrial Applications (PACIIA).
10. Queiroz-Sousa, P. O., Salgado, A. C., Pires, C. E.: A method for building personalized ontology summaries. In: Journal of Information and Data Management, 4(3), 236-236 (2013).
11. Liepins R, Cerans K, Sprogis A. Visualizing and editing ontology fragments with OWGrEd. In: CEUR Workshop Proceedings vol 932(2012), pp. 22-25.
12. Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank Citation Ranking: Bringing Order to the Web. Stanford InfoLab, (1999)