

ProGGD - Data Profiling on Knowledge Graphs using Graph Generating Dependencies

Larissa C. Shimomura^{1,*}, George Fletcher¹ and Nikolay Yakovets¹

¹*Department of Mathematics and Computer Science, Eindhoven University of Technology - Eindhoven, Netherlands*

Abstract

Data profiling is usually performed in the very first step of a data analysis pipeline. The main goal of data profiling is to present a comprehensive overview of the data contents and its properties to the user, including any potential relationships among data attributes. In knowledge graphs, the interplay among different graph entities, properties, and more complex patterns that emerge in the graph is crucial to understanding its content. In this demo paper, we introduce ProGGD, a system that employs Graph Generating Dependencies to showcase information about the graph's content.

Keywords

Data Profiling, Graph Data Dependencies, Knowledge Graphs, Graph Generating Dependencies

1. Introduction

Data profiling refers to the task of providing a comprehensive overview of the data contents and its properties to the user. To achieve this, data dependencies are used to depict potential correlations among the attributes and to express data quality rules. Such data dependencies have been extensively studied in the context of relational data [1]. In the realm of knowledge graphs, our interest extends beyond the entities' properties and attributes to also encompass information about the graph's topology. This includes the presence of specific graph patterns within the data and their possible correlations [2].

Many methods for profiling knowledge graphs have been proposed. However, few systems employ graph data dependencies to represent information about the data. The primary advantage of using graph data dependencies for profiling knowledge graphs lies in their capacity to convey detailed information about the knowledge graph to the user, and in turn, offer insights into the quality of the data.


Graph Generating Dependencies (GGDs) [3, 4] is a class of dependencies for knowledge graphs, which can, in an informal sense, express topological constraints based on two (possibly different) graph patterns and the similarity of property values of nodes and edges within those defined graph patterns. In this demo paper, we introduce ProGGD, a system designed for knowledge graph data profiling that uses GGDs to represent information about the graph.


ISWC 2023 Posters and Demos: 22nd International Semantic Web Conference, November 6–10, 2023, Athens, Greece

*Corresponding author.

✉ l.capobianco.shimomura@tue.nl (L. C. Shimomura); g.h.l.fletcher@tue.nl (G. Fletcher); n.yakovets@tue.nl (N. Yakovets)

ORCID [0009-0008-4679-7656](https://orcid.org/0009-0008-4679-7656) (L. C. Shimomura); [0000-0003-2111-6769](https://orcid.org/0000-0003-2111-6769) (G. Fletcher); [0000-0002-1488-1414](https://orcid.org/0000-0002-1488-1414) (N. Yakovets)

 © 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

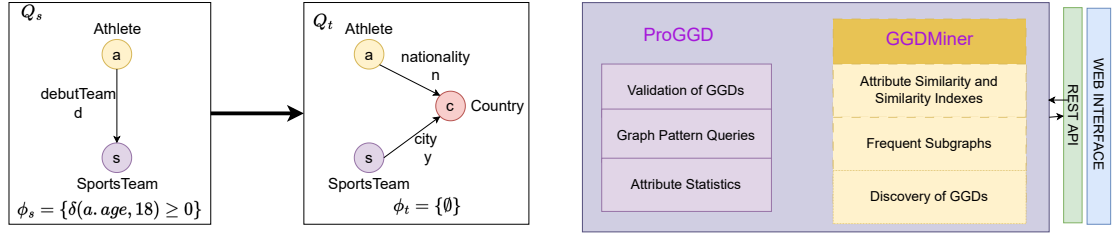


Figure 1: Example GGD enforcing the correct nationality of athletes (left). Overview of ProGGD system and functionalities (right).

Within ProGGD, given a knowledge graph, we identify GGDs from the data based on the frequency of a graph pattern’s appearance and the prevalence of similar or correlated attributes of nodes/edges in the graph. This discovery algorithm and the high expressivity of GGDs enables us to display information both at the schema level (the GGD itself) and at an instance level (data examples of each GGD). In ProGGD, our goal is to not only display interesting information about the data to the user but also make it easy to understand the discovered GGDs so that the user can also use it in their downstream tasks.

2. Proposed Approach

A GGD is defined as $Q_s[\bar{x}]\phi_s \rightarrow Q_t[\bar{x}, \bar{y}]\phi_t$ in which $Q_s[\bar{x}]$ and $Q_t[\bar{x}, \bar{y}]$ are respectively source and target graph patterns and ϕ_s and ϕ_t are respectively source and target constraints on the attributes of the nodes/edges of the respective graph patterns according to its similarity. We say that a GGD is validated in a graph G if, for every homomorphic match of the source side, there exists a homomorphic match of the target side; we refer to [4] for formal details. If a GGD is not validated, we can modify the validation algorithm to identify for which source matches the target does not exist. Observe in Figure 1 an example of a GGD which states that for every “Athlete” which has an “debutTeam” edge connected to a sports team and is aged at least 18, there should exist an edge to a “Country” node to represent their nationality and the “SportsTeam” should be located within a city of that Country, represented by the edge labelled “city”.

Given a graph G , we consider *interesting* GGDs for data profiling i.e., GGDs with graph patterns and constraints according to the similarity of the attributes (differential constraints) that: (1) occur frequently on G , (2) validated according to a user-defined rate (called *confidence* in our system) and, (3) can maximize the total number of matched nodes and edges in the graph G (called *coverage* in our system). To discover such GGDs from G , we use our GGDMiner discovery algorithm. According to these conditions, the main parameters of GGDMiner that need to be set by the user through ProGGD are frequency, confidence, and the maximum size of the result set of GGDs. Additional parameters such as the minimum threshold value for differential constraints and the maximum number of edges for graph patterns can optimize the mining process. ProGGD offer default values for these parameters to guide the users that are not familiar with the background algorithm. Due to lack of space, more information about the

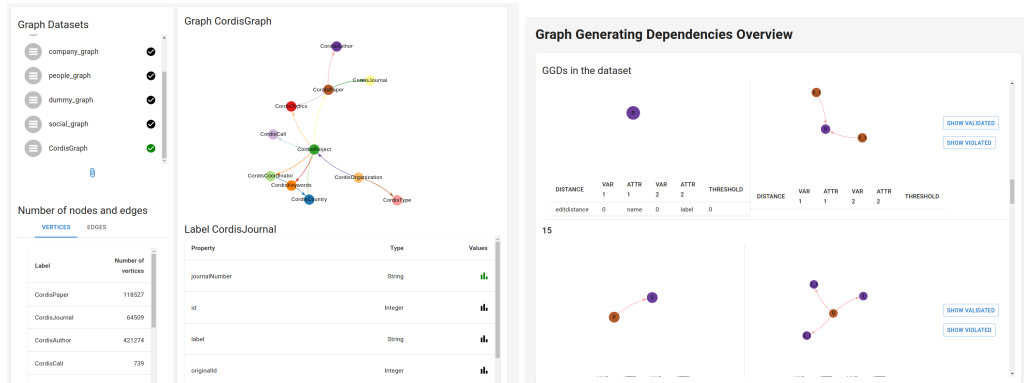


Figure 2: Screenshot of the initial panel (left) and the GGDs visualization panel (right)

GGDMiner and its parameters are available in the source code repository¹.

In a summary, the GGDMiner has the following steps. First, we identify the attributes of each node/edge label that are interesting to consider for the differential constraint discovery and construct auxiliary data structures based on the similarity of the attributes. Next, we mine graph patterns and constraints that occur frequently in G , each combination of graph pattern and set of constraints is considered a candidate for the source or target of a GGD. In this step, we use algorithms and techniques for frequent subgraph mining and discovery of association rules from the literature [5]. Finally, we verify which pairs of mined candidates can become a GGD and which of these GGDs can cover the most information about the graph G for data profiling. In each of the steps of the algorithm, we produce intermediate results which can be used to understand the knowledge graph G .

If a GGD is not validated (confidence is less than 1), it could indicate an error in the data. ProGGD allows the user to visualize examples of validated and violated graph pattern matches which might help the user to understand more about the content of the knowledge graph and why such GGD was mined. ProGGD also includes functionalities to give overall information about the graph such statistics about the attributes and graph pattern query results. We use the Spark framework as the primary backend of ProGGD, to retrieve information about the attributes and also query graph patterns by using the G-Core language [6]. Figure 1 shows an overview of the ProGGD system and the functionalities available.

3. Features and Demonstration

Because of the rich expressiveness of GGDs, ProGGD includes many functionalities. For the user's ease of navigation, we divide such functionalities into 4 main panels:(1) Metadata information and initialization, (2) Attribute information, (3) Topological Information and, (4) GGDs.

Metadata Information and Initialization - Given a knowledge graph and its schema information, in ProGGD we can visualize the schema and verify the attributes of each one of

¹<https://github.com/laricsh/ggdminer>

Dataset	Node Labels	Edge Labels	Nodes	Edges
Cordis ²	10	12	32K	151K
GDelt ³	5	2	73K	445K
DBLP ⁴	4	4	2M	810K

Table 1
Datasets used in the demonstration

the types of nodes and edges. We also display initial information about the attributes in each node/edge label to assist the user in setting the parameters for the GGD discovery algorithm. Next, we run GGDMiner to discover GGDs; this process will populate the information displayed in the other panels of the system.

Attribute Information - The GGD discovery algorithm can discover correlated attributes between the different nodes and edges of the knowledge graph based on its similarity. In this panel, we showcase possibly correlated attributes according to the semantic similarity of its property names, results of similarity join and clustering for string attributes, and the distribution of the values for numerical attributes.

Topological Information - In this panel, we display topological information according to the frequent subgraph patterns mined from the knowledge graph. In this panel, user can verify which graph patterns frequently appear and which nodes/edges are matched to each one of these graph patterns. We use an Answer Graph [7] to represent the matches of the graph patterns, which allows to visualize the matches as a subgraph efficiently, unlike systems that use table-like representation of the matches.

GGDs - In this panel, we visualize GGDs discovered from the knowledge graph along with the total number of source matches, target matches and the rate of validated sources (see screenshot in Figure 2). We also show the validated and violated matches of each GGD to help the user understand the semantics of a GGD and also which subgraphs appear correlated to each other in the knowledge graph.

For the demonstration, we use open-source graph datasets and subsets of these datasets in the context of citations networks to showcase the ProGGD functionalities. Table 1 shows details about the datasets we plan to use. During demonstration, participants can select a dataset/subset, load it into the system and explore ProGGD functionalities. We showcase ProGGD in two main scenarios. In the first scenario, we focus on the topology of the knowledge graph by exhibiting mined graph patterns and their correlations using a knowledge graph with few data attributes. The second scenario, on the other hand, demonstrates the ProGGD capabilities in terms of highlighting correlations and similarities between attributes. These two scenarios illustrate how ProGGD is useful in understanding the content of the knowledge graph, even when they differ significantly in terms of topology and the number of data attributes. More details and the source code for ProGGD are available in <https://github.com/laricsh/proggd>.

²Graph built from Horizon 2020 project information accessed on <https://data.europa.eu/data/datasets/cordish2020projects?locale=en>

³<https://github.com/smardatalake/datasets/tree/master/gdelt>

⁴<https://www.aminer.org/citation>

4. Conclusion and Future Work

In this paper, we introduced the ProGGD system, which employs GGDs for profiling knowledge graphs. The high expressivity of GGDs enables the representation of complex information about both the topology and properties that may be associated with each other in real-world knowledge graphs. ProGGD also offers the ability to inspect the matched nodes and edges of each GGD, thereby assisting users in understanding the content of their knowledge graph beyond mere metadata information. As part of our future work, we plan to improve ProGGD's scalability and include functionalities that utilize GGDs in other tasks, such as data cleaning and data integration.

Acknowledgments

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreements No. 825041 and No. 101058573.

References

- [1] Z. Abedjan, L. Golab, F. Naumann, Profiling relational data: A survey, *The VLDB Journal* 24 (2015) 557–581. doi:10.1007/s00778-015-0389-y.
- [2] M. Ben Ellefti, Z. Bellahsene, J. G. Breslin, E. Demidova, S. Dietze, J. Szymański, K. Todorov, RDF dataset profiling—a survey of features, methods, vocabularies and applications, *Semantic Web* 9 (2018) 677–705.
- [3] L. C. Shimomura, G. Fletcher, N. Yakovets, GGDs: Graph generating dependencies, in: *Proceedings of the 29th ACM International Conference on Information & Knowledge Management, CIKM '20*, Association for Computing Machinery, New York, NY, USA, 2020, pp. 2217–2220. doi:10.1145/3340531.3412149.
- [4] L. C. Shimomura, N. Yakovets, G. Fletcher, Reasoning on property graphs with graph generating dependencies, 2022. arXiv:2211.00387, *Under Review*.
- [5] M. Elseidy, E. Abdelhamid, S. Skiadopoulos, P. Kalnis, Grami: Frequent subgraph and pattern mining in a single large graph, *Proc. VLDB Endow.* 7 (2014) 517–528. doi:10.14778/2732286.2732289.
- [6] R. Angles, M. Arenas, P. Barcelo, P. Boncz, G. Fletcher, C. Gutierrez, T. Lindaaker, M. Paradies, S. Plantikow, J. Sequeda, O. van Rest, H. Voigt, G-core: A core for future graph query languages, in: *Proceedings of the 2018 International Conference on Management of Data, SIGMOD '18*, Association for Computing Machinery, New York, NY, USA, 2018, p. 1421–1432. doi:10.1145/3183713.3190654.
- [7] Z. Abul-Basher, N. Yakovets, P. Godfrey, S. Clark, M. H. Chignell, Answer graph: Factorization matters in large graphs, in: *Proceedings of the 24th International Conference on Extending Database Technology, EDBT 2021*, 2021, pp. 493–498. doi:10.5441/002/edbt.2021.57.