Graph Profiling with Graph Generating Dependencies

Larissa C. Shimomura¹

¹Supervised by Nikolay Yakovets and George Fletcher Eindhoven University of Technology, Eindhoven - Netherlands

Abstract

Data dependencies have a direct application to data profiling as they can provide information about the metadata. The rise of practical real-world use of graph data resulted in increased interest in studying dependencies and constraints in graphs and their applications. In this project, we propose a new class of dependencies for graph data named Graph Generating Dependencies, or GGDs. Informally, a GGD expresses constraints between two (possibly different) graph (sub-)structures enforcing dependencies according to user-defined topological (graph) patterns and similarities in the corresponding data (property) values. The expressiveness of GGDs allows to describe information about the graph data about both topology and property values, making it an interesting class of dependency for graph data profiling. In this paper, we present the GGDs and further topics on graph data profiling that we plan to investigate during the project

Keywords

Graph Dependencies, Data Profiling, Tuple-Generating Dependencies, Graph Databases

1. Introduction

Data profiling refers to the task of collecting information about the content of the data. In the area of data profiling, data dependencies are used as a tool not only to assist the user in understanding the data and possible correlations between different attributes but also as a tool to express and ensure data quality rules. The property graph model is the emerging standard with current efforts from both industry and academia on standardizing a graph query language (GQL)¹. Therefore, it is important to define and study new classes of dependencies for this model and its practical use.

Consider a social network graph in which it has been identified that whenever two *people* vertices have the same last name and address property values, and have an edge labelled *"friend"* connecting both, then there should also exist an edge of the type *"is family"* connecting these two vertices. It is important to be able to capture and present such constraints to the user as can arise naturally in graph data and such information is valuable for further profiling and use of the data.

However, classes of graph dependencies [1, 2, 3] for the property graph model previously proposed in the literature cannot fully capture such information as they are defined over one graph pattern and focus on generalizing functional dependencies (i.e., variations of egds, *equality*-generating dependencies). To represent such information, in this project, we propose a new class of graph dependencies for the property graph model named Graph Generating Dependencies (GGDs) [4]. A GGD can express topological constraints according to two (possibly) different graph patterns and data constraints that express the similarity of the property values of nodes and edges on the defined graph patterns. Given the expressiveness of GGDs, this class of dependencies can be used in practical scenarios such as describing the content of the data (discovery and visualization of GGDs) and ensuring data quality (detecting data inconsistencies, entity resolution and repair of graph data).

In this paper, we introduce in section 3 the syntax and semantics of GGDs, its main reasoning problems and practical use cases of GGDs. section 4 we present the topics regarding the use of GGDs for graph data profiling and exploration that we are currently investigating.

2. Related Work

We place GGDs in the context of relational and graph dependencies proposed in the literature.

The classical Functional Dependencies (FDs) have been studied and extended for contemporary applications in data management. Conditional Functional Dependencies (CFDs) [5] were later proposed for data cleaning tasks. CFDs enforce an FD only for a set of tuples specified by a condition, unlike the original FDs, in which the dependency holds for the whole relation. Due to its large application to data cleaning, discovery algorithms and extensions were proposed for CFDs [6].

The idea of FDs and CFDS were extended to graph dependencies on Graph Functional Dependencies (GFDs) [1] and Graph Entity Dependencies (GEDs) [2].

Proceedings of the VLDB 2022 PhD Workshop, September 5, 2022. Sydney, Australia. Copyright (C) 2022 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

[▲] l.capobianco.shimomura@tue.nl (L.C. Shimomura)

Original Cellson (CEUR Workshop Proceedings (CEUR-WS.org)

¹https://www.gqlstandards.org/home

The GFDs are formally defined as a pair $(Q[\bar{x}], X \rightarrow Y)$ in which $Q[\bar{x}]$ is a graph pattern that defines a topological constraint, and X, Y are two sets of literals over the vertex attributes, that define the property-value functional dependencies of the GFD. Besides the property-value dependencies present in the GFDs, GEDs also carry special id literals to enable identification of vertices in the graph pattern.

Differential Dependencies (DDs) [7] were proposed to support applications such as entity resolution, in which the similarity of the attributes must be considered. The DDs extend the FDs by specifying constraints according to user-defined distance functions between attribute values [7]. This idea was also introduced in a class of graph dependencies, the Graph Differential Dependencies - GDDs [3].

Recently, PG-Keys were proposed as a formalism to define keys over property graphs [8]. While GFDs and GEDs define constraints only over vertex attributes, PG-Keys can identify and constraint unique vertices, edges and properties in the property graph.

Tuple-generating dependencies (tgds) are a wellknown type of dependency used in the areas of data integration and data exchange [9]. Special cases of tgds and its extensions also have a wide range of applications. One example of a special case of tgds are inclusion dependencies (INDs) that are often used to identify foreign keys in relational data [10]. Close to the idea of having constraints for property values, there is an extension to egds and tgds called constrained tuple-generating dependencies (ctgds) [11]. The ctgds extends the tgds by adding a condition (a constraint) on variables.

Other types of constraints for graphs include: Graph Repairing Rules (GRRs [12]), an automatic repairing semantics for graphs, and Graph-Pattern Association Rules (GPARs [13]), a specific case of tgds and has been applied to social media marketing applications.

The main differences of GGDs compared to previous works are: (i) the use of differential constraints, (ii) edges are first-class citizens in the graph patterns (in alignment with the property graph model) and (iii) the ability to entail the generation of new vertices and edges. In general, GGD is the first constraint formalism for property graphs supporting both egds and tgds, and DDs for property values.

3. GGDs

In this section, we present the Graph Generating Dependencies (GGDs), which includes: GGDs definition, reasoning problems and examples of practical use cases of GGDs.

3.1. Syntax and Semantics of GGDs

We start by presenting the syntax and semantics of the GGDs, previously published in [4]. A **Graph Generating Dependency** (GGD) is a dependency of the form

$$Q_s[\overline{x}], \phi_s \to Q_t[\overline{x}, \overline{y}], \phi_t$$

where:

- Q_s[x̄] and Q_t[x̄, ȳ] are graph patterns, called source graph pattern and target graph pattern, respectively;
- ϕ_s is a set of differential constraints defined over the variables \overline{x} (variables of the graph pattern Q_s); and
- ϕ_t is a set of differential constraints defined over the variables $\overline{x} \cup \overline{y}$, in which \overline{x} are the variables of the source graph pattern Q_s and \overline{y} are any additional variables of the target graph pattern Q_t .

A differential constraint in ϕ_s on $[\bar{x}]$ (resp., in ϕ_t on $[\bar{x}, \bar{y}]$) is a constraint of one of the following forms [3, 7]:

1.
$$\delta_A(x.A, c) \le t_A$$

2. $\delta_{A_1A_2}(x.A_1, x'.A_2) \le t_{A_1A_2}$
3. $x = x'$ or $x \ne x'$

$$5. x = x \quad 01 \ x \neq x$$

where $x, x' \in \overline{x}$ (resp. $\in \overline{x} \cup \overline{y}$) for $Q_s[\overline{x}]$ (resp. for $Q_t[\overline{x}, \overline{y}]$), δ_A is a user defined similarity function for the property A and x.A is the property value of variable x on A, c is a constant of the domain of property A and t_A is a predefined threshold. The differential constraints defined by (1) and (2) can use the operators $(=, <, >, \leq, \geq, \neq)$.

The constraint (3) x = x' states that x and x' refer to the same entity (vertex/edge) and can also use the inequality operator stating that $x \neq x'$. An important feature of GGDs is that both vertices and edges are considered variables (in source and target graph patterns), which allows the comparison of vertex-vertex variables, edge-edge, and vertex-edge variables.

Consider a graph pattern $Q[\overline{z}]$, a set of differential constraints ϕ_z and a match of this pattern represented by $h[\overline{z}]$ in a graph *G*. The match $h[\overline{z}]$ satisfies ϕ_z , denoted as $h[\overline{z}] \models \phi_z$ if the match $h[\overline{z}]$ satisfies every differential constraint in ϕ_z . If $\phi_z = \emptyset$ then $h[\overline{z}] \models \phi_z$ for any match of the graph pattern $Q[\overline{z}]$ in *G*.

A GGD $\sigma = Q_s[\bar{x}], \phi_s \rightarrow Q_t[\bar{x}, \bar{y}], \phi_t$ holds in a graph G, denoted as $G \models \sigma$, if and only if for every match $h_s[\bar{x}]$ of the source graph pattern $Q_s[\bar{x}]$ in *G* satisfying the set of constraints ϕ_s , there exists a match $h_t[\bar{x}, \bar{y}]$ of the graph pattern $Q_t[\bar{x}, \bar{y}]$ in *G* satisfying ϕ_t such that for each *x* in \bar{x} it holds that $h_s(x) = h_t(x)$. In case a GGD does not hold in *G* (it is violated), it can be repaired by *generating* new vertices/edges in *G*.

Example - The GGD in Figure 1 describes that whenever there is a match of the source graph pattern, in which



Figure 1: Example of a GGD

a Person node of the type "TV-Personality" is connected to Magazine by the edge "appeared on", there should exist an edge of the type "is about" from this Magazine to a node labeled Genre in which its attribute name is "TV-Shows".

3.2. Reasoning of GGDs

To understand the application of GGDs in real-world data and its properties, we study how we can reason about GGDs. We discuss three reasoning problems for GGDs: Satisfiability, Implication, and Validation. Due to space limitation, in this section, we give an overview of how we can solve each one of these problems.

Satisfiability - A set of GGDs Σ is satisfiable if there exists a model that is a graph *G*, such that (i) $G \models \Sigma$ and (ii) for each GGD $\sigma \in \Sigma$ there exists a match of $Q_s[\overline{x}]$ in *G*. The satisfiability problem is to answer if given a set Σ , is Σ satisfiable?

Informally, the Satisfiability problem is to verify if the set of GGDs Σ is consistent. A set of GGDs Σ is not satisfiable when contradictory constraints are enforced to the same node or edge in a graph G. Similar to the Satisfiability checking for GEDs [13], the satisfiability problem for GGDs can be solved by using a Chase procedure for GGDs. Given a graph G_{Σ} which contains a match of each source side of each GGD in Σ , if the Chase procedure terminates and there are no infeasible/contradictory differential constraints enforced in any properties of any nodes/edges of G_{Σ} , we can conclude that the set Σ is satisfiable.

Implication - Given a set of GGDs Σ and a GGD σ , does Σ imply σ , (denoted by $\Sigma \models \sigma$) for every non-empty graph G that satisfies Σ ?

The implication of data dependencies can be proven by using Chase. Given a initial graph $G_{closure}$ which contains a match of the source of σ , the Chase procedure will interactively apply GGDs of Σ and enforce its target constraints. After Chase terminates, if for every match of the source of σ there exists a match of its target in $G_{closure}$ then the implication is true, otherwise, it is false.

Validation - Given a set of GGDs Σ and a non-empty graph *G*, does the set of GGDs Σ hold in *G*, denoted as $G \models \Sigma$?

The validation problem can be solved by an algorithm with the following steps:

- 1. Check if $h_s(\overline{x})$ satisfies the source constraints (i.e., $h_s(\overline{x}) \models \phi_s$). If yes then continue.
- 2. Retrieve all matches $h_t(\overline{x}, \overline{y})$ of the target graph pattern $Q_t[\overline{x}, \overline{y}]$ where $h_s(x) = h_t(x)$ for all $x \in \overline{x}$. If there are no such matches of the target graph pattern, return false.
- 3. Verify if $h_t(\bar{x}, \bar{y}) \models \phi_t$. If there exists at least one match of the target graph pattern such that $h_t(\bar{x}, \bar{y}) \models \phi_t$, then return true, else return false.

3.3. Practical Use of GGDs

In this section, we present how GGDs can be used in practice in two different scenarios: (1) Identifying data inconsistencies and (2) Entity Resolution. The algorithms used in these scenarios were implemented in the sHINER² system using the G-Core language interpreter³[14] and the Spark framework⁴.

Identifying Data Inconsistencies - Given a set of GGDs Σ , we define as inconsistent data a set of graph pattern matches of the source side of each GGD in Σ in which there does not exist a match of the target side that satisfies the target constraints ϕ_t .

From the definition of inconsistent data, we can observe that this problem is related to checking if a set of GGDs, Σ , is violated or not. For this reason, to identify inconsistent data, we modify the previously introduced Validation algorithm to return which matches of the source $(h_s(Q_s[\overline{x}) \models \phi_s))$ were not validated instead of returning true or false if the set is validated or not.

We implemented two versions of this algorithm using "left anti joins" and "left outer joins" to identify data inconsistencies. This choice of operators were based on previous studies on validation over tgds of the literature in the scenario of validating schema mappings [15, 16]. Although there is room for optimization and improvement on the implementation, the goal of these results is to show how GGDs are feasible even when using an available query engine such as SparkSQL.

We compared these two versions of the algorithms using the LDBC benchmark dataset⁵. For both of these datasets, we manually defined a set of GGDs. Even though both versions of the algorithms finished in a feasible time, the the algorithm that uses left anti join performed better in terms of scalability (see Figure 2).

Entity Resolution - Entity Resolution is the task of identifying instances of data that refer to the same real-world entity. Entity Resolution can be used not only to

²https://github.com/smartdatalake/gcore-spark-ggd

³https://ldbcouncil.org/gcore-spark/

⁴https://spark.apache.org/

⁵https://ldbcouncil.org/benchmarks/snb/



Figure 2: Scalability of Validation of GGDs



Figure 3: Example of a GGD used for Entity Resolution

deduplicate data but also to integrate datasets according to real-world entities that they have in common.

GGDs can be used in practice to describe rules for Entity Resolution. By using GGDs, we are able to declare deduplication rules over graph data according to graph patterns and the similarity of the attributes of nodes and edges defined in these graph patterns. Observe in Figure 3 an example of a GGD for Entity Resolution.

To check if two entities match according to a GGD, we can use the same modified version of the Validation algorithm used for data inconsistencies. This algorithm will identify which matches of the source graph pattern should have an edge indicating that they are the same but do not actually have one in the dataset. Given this set of matches, the next step is to generate the new edges. We implemented a version of the GGDs repair algorithm by assuming that these missing edges will always be generated. Since the generation of an edge can trigger the validation of another GGD, this algorithm stops when there are no changes in the graph.

Using our implementation in sHINER and the expertise of our industrial partners in the SmartDataLake⁶ project, we set the GGDs manually and tested their performance on datasets from our industrial partners. Our partners were able to achieve the same Recall and Precision using GGDs compared to their internal tools, however, with less human effort to run entity resolution.

⁶https://smartdatalake.eu/

4. Research Plan

Given the work on GGDs, in this section we introduce the topics that we will investigate on GGDs for graph data profiling.

Approximate Discovery of GGDs - Discovering data dependencies from the data is not a trivial task, and new challenges arise when dealing with graph data. Dependency classes proposed for graph data in the property graph model are usually defined over a graph pattern, which means that not only the constraints over attributes should be discovered, but also the constraints over the topology must be identified by the discovery algorithms. Discovery algorithms proposed for graph dependencies [1, 17] uses frequent graph pattern mining algorithms to find the most relevant graph patterns. However, the discovery of GGDs is even more challenging as correlations between graph patterns and the similarity of the attributes/properties of its nodes and edges should be identified. To the best of our knowledge, there is no method that can discover such correlations. Our goal is to develop an algorithm in which given a graph data Gand a degree of inconsistency τ outputs a set of GGDs Σ which holds on *G* with at most τ degree of violation. To solve such problem we are currently working on an approach that uses the ideas proposed in the association rule mining area [18] to discover correlations between graph patterns, and on metrics or strategies that can quantitatively measure τ .

Visualization of GGDs - One of the main advantages of dependencies is that their syntax is human interpretable. However, depending on the content and volume of the data, it can be difficult for the general user to understand the cases in which a dependency holds or not. While there are many tools that have explored the idea of visualization of data dependencies using relational data [19], to the best of our knowledge, this topic has not been explored in the context of graph data. Thus, many of these systems focus on ways to visualize the semantics of the dependency and not how this dependency occurs in the data. Given these challenges, the goal is to develop a system in which the user can understand the GGDs through visualization of examples of data in which the GGDs hold.

User-guided repair using GGDs - Repairing graphs with constraints is a key task to ensure data quality. The repairing problem for GGDs can be defined as given a set of GGDs Σ and a data graph *G*, make $G \models \Sigma$, meaning Σ holds on *G*. Given the "generating" property of GGDs, a naive way to repair the graph data is to always create new nodes or edges. However, this solution can create even more noise in the data and might not generate useful information. To avoid this situation, the knowledge of the dataset specialist is crucial to correctly clean the data [20]. Involving the user can be very expensive because of the

large number of possibilities to be verified. For this reason, this topic has two main challenges: (1) develop a mechanism to suggest the best option on how to repair the data to the user and; (2) develop a policy on how and which order the suggestions should be presented to the user. To the best of our knowledge there is no method in user-guided repair using dependencies in the property graph model, however, studies in the context of relational data and on knowledge bases [20, 21] can be reviewed and provide inspiration to solve the problem.

5. Conclusion

In this project, we propose GGDs, a new class of dependencies for the property graph model. GGDs can express an association between (possibly different) graph patterns and their attributes. GGDs can be used to describe meaningful information about the graph data and assist the user in further data analysis. In this paper, we presented the definition of GGDs, practical use cases of GGDs and the topics and challenges that we will investigate on graph data profiling using GGDs.

Acknowledgments

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

References

- W. Fan, Y. Wu, J. Xu, Functional dependencies for graphs, in: SIGMOD, 2016, pp. 1843–1857.
- [2] W. Fan, P. Lu, Dependencies for graphs, ACM Trans. Database Syst. 44 (2019) 5:1–5:40.
- [3] S. Kwashie, L. Liu, J. Liu, M. Stumptner, J. Li, L. Yang, Certus: An effective entity resolution approach with graph differential dependencies (gdds), Proc. VLDB Endow. 12 (2019) 653–666.
- [4] L. C. Shimomura, G. Fletcher, N. Yakovets, Ggds: Graph generating dependencies, in: Proceedings of the 29th ACM International Conference on Information amp; Knowledge Management, CIKM '20, Association for Computing Machinery, New York, NY, USA, 2020, p. 2217–2220.
- [5] P. Bohannon, W. Fan, F. Geerts, X. Jia, A. Kementsietsidis, Conditional functional dependencies for data cleaning, in: ICDE, 2007, pp. 746–755.
- [6] W. Fan, F. Geerts, J. Li, M. Xiong, Discovering conditional functional dependencies, IEEE Transactions on Knowledge and Data Engineering 23 (2011) 683–698.

- [7] S. Song, L. Chen, Differential dependencies: Reasoning and discovery, ACM Trans. Database Syst. 36 (2011) 16:1–16:41.
- [8] R. Angles, A. Bonifati, S. Dumbrava, G. Fletcher, K. W. Hare, J. Hidders, V. E. Lee, B. Li, L. Libkin, W. Martens, F. Murlak, J. Perryman, O. Savković, M. Schmidt, J. Sequeda, S. Staworko, D. Tomaszuk, Pg-keys: Keys for property graphs, in: Proceedings of the 2021 International Conference on Management of Data, SIGMOD/PODS '21, Association for Computing Machinery, New York, NY, USA, 2021, p. 2423–2436.
- [9] R. Fagin, P. G. Kolaitis, R. J. Miller, L. Popa, Data exchange: Semantics and query answering, in: D. Calvanese, M. Lenzerini, R. Motwani (Eds.), Database Theory – ICDT 2003, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, pp. 207–224.
- [10] J. Bauckmann, U. Leser, F. Naumann, V. Tietz, Efficiently detecting inclusion dependencies, in: 2007 IEEE 23rd International Conference on Data Engineering, 2007, pp. 1448–1450. doi:10.1109/ICDE. 2007.369032.
- [11] M. J. Maher, D. Srivastava, Chasing constrained tuple-generating dependencies, in: Proceedings of the fifteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, 1996, pp. 128–138.
- [12] Y. Cheng, L. Chen, Y. Yuan, G. Wang, Rule-based graph repairing: semantic and efficient repairing methods, in: ICDE, 2018, pp. 773–784.
- [13] W. Fan, Dependencies for graphs: Challenges and opportunities, J. Data and Information Quality 11 (2019) 5:1–5:12.
- [14] 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, Gcore: 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.
- [15] B. Alexe, B. ten Cate, P. G. Kolaitis, W.-C. Tan, Designing and refining schema mappings via data examples, in: Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data, SIGMOD '11, Association for Computing Machinery, New York, NY, USA, 2011, p. 133–144.
- [16] A. Bonifati, G. Mecca, A. Pappalardo, S. Raunich, G. Summa, Schema mapping verification: The spicy way, in: Proceedings of the 11th International Conference on Extending Database Technology: Advances in Database Technology, EDBT '08, Association for Computing Machinery, New York, NY, USA, 2008, p. 85–96.
- [17] M. Alipourlangouri, F. Chiang, Keyminer: Discov-

ering keys for graphs, in: VLDB workshop TD-LSG, 2018.

- [18] W. Fan, X. Wang, Y. Wu, J. Xu, Association rules with graph patterns, Proc. VLDB Endow. 8 (2015) 1502–1513.
- [19] B. Breve, L. Caruccio, S. Cirillo, V. Deufemia, G. Polese, Visualizing dependencies during incremental discovery processes., in: EDBT/ICDT Workshops, 2020.
- [20] M. Yakout, A. K. Elmagarmid, J. Neville, M. Ouzzani, I. F. Ilyas, Guided data repair, Proceedings of the VLDB Endowment 4 (2011).
- [21] A. Arioua, A. Bonifati, User-guided repairing of inconsistent knowledge bases, in: EDBT: Extending Database Technology, OpenProceedings. org, 2018, pp. 133–144.