Heterogeneity Detection and Comparison for Cost-Aware Graph Schema Evolution and Transformation

Extending Graph Databases with Data Profiles to Determine Heterogeneity-Based Cost Functions

Dominique Hausler[1,*], Meike Klettke[1]

[1]University of Regensburg, Data Engineering Group, Regensburg, Bavaria, Germany

Abstract

Graph databases allow direct data insertion without predefining a schema. While flexible, this often leads to a relaxed schema, especially without the usage of schema checking component, such as constraints. Consequently, heterogeneous structures emerge, requiring identification during data exploration. Furthermore, evolving such a relaxed schema can increase this heterogeneity even more. In this paper, we present Graph Data Profiles (GDPs) which capture initial heterogeneity by extracting statistical metadata (e.g., the number of labels and property keys for each node and edge). Additionally, these include uniqueness constraints (i.e., unique column combinations), existence constraints (mandatory or optional properties), missing edges, type constraints and valid inclusion dependencies. To support schema evolution and transformation, we introduce the GDP Diff, which underlines changes in heterogeneity. Thereby, inclusion dependencies help to detect in-version redundancies and evolution operations such as renaming or moving between versions. In contrast to other works, we address both schema evolution – essential for aligning a schema with changing requirements – and schema transformation, an frequent task in database monitoring. Our approach includes a cost calculation based on (1) a formalization of evolution or transformation operations and (2) heterogeneity metrics derived from the GDPs. Subsequently, GDPs ensure the detection of heterogeneity during data exploration, while the GDP Diff highlights the impact of an operation on the schema. This ensures user-informed decision making, supporting users to effectively clean unintended heterogeneity, such as typos.

Keywords

Schema Evolution, Schema Transformation, Graph Data Profiling, Heterogeneity, Inclusion Dependencies, Constraints, Cost Functions

1. Introduction

Schema evolution is essential to retain the functionality of a system with changing external and internal requirements. Thereby, even simple evolution operations can cause complex underlying schema changes, especially when multiple software engineers operate on a single graph database. Furthermore, users cannot estimate whether an evolution operation (`add`, `rename`, `delete`, `copy`, `move`, `split`, `merge`) affects all selected data, a subset of the data, or none at all. Since schema-less databases allow direct population without a predefined schema, issues regarding heterogeneous structures such as typos frequently arise.

Both schema evolution and transformation share identical components but are distinguished by which component needs to be derived. Evolutionary tasks take a source schema and evolution operations to generate a target schema. In contrast, schema transformation compares a source and a target schema to identify schema modifications. We envision a modular framework that integrates both scenarios, aiming to detect, visualize, and estimate the impact of schema evolution and transformation on heterogeneity.

To address the issue of relaxed schemata, for instance, in collaborative environments, we present *Graph Data Profiles* (GDPs) to capture heterogeneity in an initial data exploration phase by extracting statistical metadata (e.g. number of occurrences for labels) and different constraints (e.g., uniqueness,

Published in the Proceedings of the Workshops of the EDBT/ICDT 2026 Joint Conference (March 24-27, 2026), Tampere, Finland

*Corresponding author.

dominique.hausler@ur.de (D. Hausler); meike.klettke@ur.de (M. Klettke)

0009-0004-2381-133X (D. Hausler); 0000-0003-0551-8389 (M. Klettke)

inclusion dependencies). Besides illustrating and quantifying heterogeneity, the GDPs enable user-informed decisions during schema evolution or version comparison (schema transformation). More precisely, they underline potential syntactic and semantic errors. These insights allow users to align the schema with their intentions by cleaning accidental errors with our tool NAUTILUS. Thus, our approach offers a modular solution for data exploration, schema evolution, and transformation.

To compare changes in irregularity a *Graph Data Profile Diff* (GDP Diff) is created (see Table 1), enabling users to track how the degree of heterogeneity changed through a preview option during schema evolution. In terms of transformation, the Diffs give insights into how heterogeneity was modified between versions. Additionally, a formalization for each evolution operation (see Table 2) is provided. These formal definitions are combined with the heterogeneity metrics stored in the GDPs to estimate an operation's complexity. This is essential, as heterogeneity substantially impacts the costs of schema changes.

Contribution We propose GDPs to display heterogeneity in schema-less graph databases by extracting statistical metadata to make implicit structures explicit during data exploration. Moreover, the GDPs incorporate constraints, namely uniqueness, existence, type constraints, and inclusion dependencies. Users benefit from GDP Diffs when monitoring pipelines, as they depict changes in the degree of heterogeneity between versions. During schema evolution, a preview of the emerging GDP Diff alongside the operation's cost is displayed, illustrating its impact. Inclusion dependencies, hereby, support an automatized detection of adjustments, caused by rename or move operations, and in-version redundancies emerging when copying. Due to the modularity of our framework, it is applicable for both schema evolution and transformation.

2. Related Work

Schema Evolution Schema Evolution is defined as the process of modifying a schema to align with new requirements. Due to its importance, it is subject in a variety of works [1, 2, 3, 4]. Even though graph databases are schema-less, PG-Schema resembles the state-of-the-art schema description language [5, 6] for which constraints are defined in [7, 8]. [9] and [10] present approaches on schema extraction, considering challenges emerging in schema-flexible property graphs. A model-driven approach for schema evolution is presented in [11]. Extending these, we present a modular framework to detect initial irregularities through GDPs alongside GDP Diffs, aiding the estimation of an operation's impact and allowing a heterogeneity comparison between versions. Depicting heterogeneity supports user-informed decisions, e.g. to clean relaxed structures.

Schema Transformation Schema transformation is a generalization of schema evolution, and can take place between two arbitrary schemata or databases. It requires schema matching (to identify differences between a source and target schema) and mapping strategies (to transform a source database into a target databases). Various matching techniques are categorized in [12]. In [13] schemata are converted into graphs, followed by graph matching methods. Pattern matching approaches are shown in [14, 15, 16]. [17] analyzes transformation between RDF and property graphs. Rafe et al. [18] present a graph meta model to represent semantics and estimate changes via graph transformation rules. Andersen et al. [19] automatically detect a minimal rule set, describing graph modifications. Our framework compares the GDPs extracted from source and target database and generates GDP Diffs, emphasizing the aspect of heterogeneity. We formalize evolution and transformation operations, to be later incorporated in our cost model.

Graph Data Profiling Data profiling aims to gain insights into a dataset. In [20] renown profiling strategies are adapted to graph data and categorized systematically. In alignment with this work, we extract structural metadata, such as the number of occurrences, and constraints, such as uniqueness and existence, to be integrated in our GDPs during data exploration. Work on graph dependencies

is presented in [21] and [22]. We, instead, integrate distance and similarity metrics often used to detect inclusion dependency candidates. Moreover, we plan on integrating our bottom-up inclusion dependency search [23] to (1) detect identical in-version data, resulting from `copy` or `move` operations [24], and (2) to automatically identify `rename` or `move` operations during schema transformation.

Cost Functions Cost functions are used to quantify and estimate the complexity of a task or operation. de Oliveira Werneck et al. [25] present a learning-approach for graph matching tasks. [26] calculates costs for sequential path optimization, while Chen et al. [27] aim to improve multi-level logic optimization algorithms. To determine similarities between graph patterns, Neuhaus and Bunke [28] show how cost functions can be learned automatically. The metrics stored in the GDPs as well as the formalized operations are integrated in our cost model, which aim to foresee the complexity and effect of an operation.

Data Quality Marchesin et al. [29] highlight the importance of data quality in Knowledge Graphs (KGs). This results from the integration of LLMs to enrich KG environments, making an automatized data quality evaluation necessary to ensure reliability. To increase the acceptance of AI systems used for classification tasks, insights into data quality are essential [30]. [31] conducts a review on data quality dimensions, whereas [32] summarizes available tools. We include the concept of GDPs to emphasize heterogeneity, aiming for user-informed decisions to clean unintentionally emerged, relaxed structures by executing evolution operations. These are defined in our evolution language GEO [33], intuitively describing an operation's effect, thus, widening the range of users to non-domain experts.

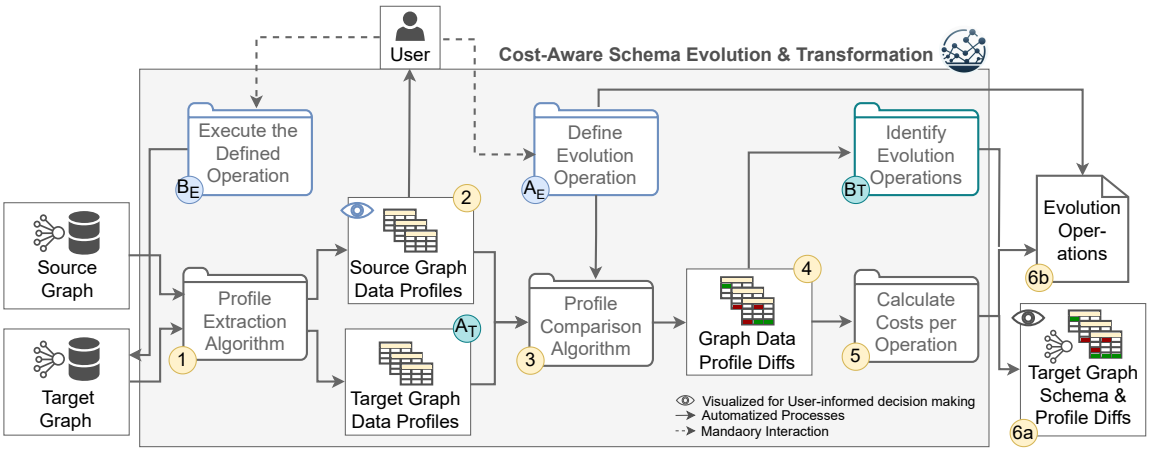

Figure 1: Workflow of our framework illustrating schema evolution (blue, *E* index), transformation (turquoise, *T* index) and shared elements (gray). Forwarded elements (eye symbol) ensure user-informed decision making to increase data quality.

3. Use Cases

Typical use cases in which our framework is beneficial:

Use Case 1: In-Version Heterogeneity In a medical company, multiple software engineers simultaneously manage a graph database without schema constraints, leading to irregularities. To detect these variations, a data analyst uses our GDPs to clean unintentionally created variations by defining evolution operations in our evolution language GEO. Subsequently, data quality is improved by homogenizing the schema in alignment with user needs.

Use Case 2: Heterogeneity Between Versions In the second scenario, a data analyst compares different database versions. Our framework thereby, automatically generates GDP Diffs to identify transformation operations including their costs. This allows the analyst to quantify whether heterogeneity increased e.g. by adding new data variants or decreased through homogenization. Understanding schema changes when monitoring or optimizing pipelines is essential, for instance to draw conclusions for similar pipelines in the manner of a knowledge base.

Interim Conclusion Both scenarios outline how error-prone schemata in collaborative environments are. Thus, illustrating the need to detect heterogeneous structures through our GDPs. Moreover, GDP Diffs ensure the estimation of an evolution operation's impact and costs. During monitoring tasks, the framework assists users by automatically detecting transformation operations, reducing manual adjustments and overcoming a missing documentation of the evolutionary process.

4. System Overview

Figure 1 shows the workflow of our envisioned, modular framework. Blue elements indexed with E are schema evolution specific, turquoise elements marked with T illustrate schema transformation, whereby gray components are part of both tasks. In *Step 1*, a source graph database serves as input, followed by the extraction of the schema components required to generate GDPs in *Step 2*. For schema transformation, the GDPs are extracted and generated for both the source and target graph data (*Step* A_T). In contrast, for schema evolution, the GDPs are visualized to the user – indicated by the eye icon – to depict in-version heterogeneity, i.e., synonyms or typos. Based on the information on heterogeneous structures in the GDPs, the user can define an evolution operation (*Step* A_E), for instance, to decrease heterogeneity by renaming labels with typos. Thereafter, the GDP Diffs are generated in *Step 3* and displayed in *Step 4*. In *Step 5*, the metrics stored in the GDPs and the formalized operations (see Table 2) are incorporated in the cost functions to illustrate the complexity of evolution or transformation operations. The output consists of an interactive schema visualization and the GDP Diffs for each entity, i.e., nodes and edges, (*Step 6a*) together with an evolution or transformation operation log-file (*Step 6b*). During schema evolution, the user then can decide whether the operation is executed (*Step* B_E) or not. Moreover, the GDP Diffs incorporate inclusion dependencies to automatically resolve ambiguities when comparing versions in *Step* B_T.

5. Framework

We present a modular framework that extends the graph evolution tool Nautilus – implementing our evolution language Geo (**G**raph **E**volution **O**peration) [34] – with Graph Data Profiles (GDPs). These depict heterogeneous graph data and form the basis for predicting an operation's effort.

Heterogeneity Presumptions As a result of the modeling freedom offered by schema-less databases, we distinguish between three ways of storing graph data. (1) Analogous to relational databases, users can predefine a non-relaxed schema in a schema-first approach, utilizing constraints. (2) Just like in *Use Case 1*, the data can be directly inserted without using schema constraints. (3) The third scenario is a partially schema controlled usage, where homogeneous structures are forced by constraints, while heterogeneous parts are kept schema-free. If schema-first approaches allow relaxed structures like optional properties, they also belong to this class. Our approach focuses on the third case, as it is often unclear whether observed heterogeneity is intentional. Hereby, our GDPs support users in homogenization – for instance, by enforcing existence on formerly optional properties – and increasing heterogeneity – e.g., by adding variants or evolving subsets. This allows users to choose between homogeneity of data and structural variations in the data when evolving graph data. Additionally, the GDP Diffs give insight into heterogeneity modifications during version comparison like in *Use Case 2*.

✗ Customer → ↑ **Client**

Labels

Label	Occurrence		Syntactic & Semantic Metrics		Constraints
	∑	%	Levenshtein	Wu-Palmer	IND
✗ Customer → ↑ Client	65 → ↑ 90	100.00%	customer, ✗ Cusotmer	✗ Client	✗ Cusotmer
KeyAccount	23	35.38% → ↓ 25.56%	Account	–	–

Properties

Property Key	Occurrence		Syntactic & Semantic Metrics		Constraints				Super-Type
	∑	%	Levenshtein	Wu-Palmer	Uniqueness	Existence	Datatype	IND	
first_name	65 → ↑ 82	100.00% → ↓ 91.12%	✗ Cusotmer.fiRst_name		–	–	✓	✗ Cusotmer.fiRst_name	✗ Customer → ↑ Client
+ firstName	3	3.34%			–	–	+		↑ Client
✗ last_name → + name	65 → ↑ 90	100.00%	customer.lastName	✗ Client.name	–	✓	✓	–	✗ Customer → ↑ Client
cust_number	65 → ↑ 90	100.00%	–		✓	✓	+	Product.id	✗ Customer → ↑ Client
member	23	35.38% → ↓ 25.56%	–	–	–	–	–	–	KeyAccount

Associated Edges

Label	Occurrence		Syntactic & Semantic Metrics		Constraints		Edge-Specifics		
	∑	%	Levenshtein	Wu-Palmer	Existence	IND	direction	start label	end label
HAS	67	100.00%	HAS	–	✓	–	ingoing	Store	–
BOUGHT	60 → ↑ 84	92.31% → ↑ 93.33%	–	–	–	✗ BOUghT	outgoing	–	Product
✗ BOUGHT	6	8.96%	–	–	–	✗ BOUghT	outgoing	–	produtc
+ HAS	48	53.33%	–	–	–	–	outgoing	–	Membership

Table 1
Graph Data Profile Diff for former `Customer` nodes, renamed to `Client`. Highlighted rows show added (green) and removed (red) items, ↑ and ↓ mark number and percentage changes along with + and ✗ marking other adjustments, e.g., created constraints or removed variants.

Graph Data Profiles (GDPs) To make heterogeneous structures explicit as needed in *Use Case 1*, we present GDPs incorporating statistical and structural metadata – extracted by the algorithm used in *Step 1* of Figure 1 – for each label of a given graph database. The term *label* is used for both node labels and edge types. Since relaxed schemata are common in collaborative environments, detecting irregularities during data exploration provides deep insights into the dataset's structure. Emphasizing heterogeneity additionally depicts candidates for data cleaning, thereby enabling the analyst to optimize the schema in alignment with data quality aspects. Distance metrics display syntactic and semantic in-version candidates, whereby valid inclusion dependencies aid identifying identical information. Furthermore, constraints highlight homogeneous structures that could be schema-controlled.

GDP Diff Table 1 depicts an exemplary GDP Diff, serving as a preview before executing an evolution operation. Consequently, supporting user-driven decision making by depicting schema changes in the context of heterogeneity. Such a preview is also beneficial to perform data cleaning (*Use Case 1*). When monitoring pipelines, the GDP Diffs facilitate a deeper understanding of schema modification and each operation's effect on the dataset's heterogeneity, as described in *Use Case 2*.

The GDP Diff in Table 1 illustratively shows `Client` nodes, formerly labeled `Customer`. The ✗ in the header shows that the label `Customer` was deleted. Changes in structure metadata such as occurrence are icon-coded by ↑ for increasing and ↓ for decreasing numbers. 25.56% of the `Client` nodes are additionally labeled `KeyAccount`, i.e., depicting multi-labeling. The Diff shows a decreasing number of multi-labeled `Clients`, caused by increasing their number from 65 to 90. Furthermore, syntactic and semantic heterogeneity metrics – namely Levenshtein to detect typos and Wu-Palmer for synonyms – are shown to detect structural irregularities. Precisely, the variation `customer` was identified for the `Client` label.

In order to analyze properties, constraints need to be made explicit. This includes deriving uniqueness constraints, resembling unique column combinations, together with existence and type constraints from the data. Moreover, valid inclusion dependencies serve an in-version detection of identical data, caused by `copy` operations, as it is the case for $Customer.first\_name \subseteq Cusotmer.fiRst\_name$. Here, the GDP Diff illustrates that this error was later cleaned by ✗ under Levenshtein. A relaxed candidate generation gives insights into renamed or moved elements (syntactic and semantic metrics). In terms of inclusion dependencies, false positives might occur, e.g., $Client.cust\_number \subseteq Product.id$. Nevertheless, user-centric cleaning decisions ensure robustness, while depicting potential overlaps. Another key aspect is illustrated in the *super-type* cell. Here the associated label for each property is depicted. Solely for node entities all associated edges are displayed, with the objective of identifying syntactic and semantic errors, as it is the case for ✗ BOUGHT, as well as to display missing and required edges ($\widehat{=}$ existence constraint).

Based on the GDP Diff, the following representative conclusions can be drawn: The deletion of the associated edge labeled BOUGHT, formerly connecting `Customer` and `produtc`, a variant of the `(Customer)-[BOUGHT]→(Product)` pattern, represents a homogenization action. Consequently, a GDP Diff-driven analysis reveals a renaming process. In the *properties* section the key `firRst_name` from `Cusotmer` was added, resembling a `move` operation.

Cost Functions Besides GDPs, we envision a cost model to capture the impact of each operation executed during schema evolution or transformation. In this context, heterogeneity can cause severe structural changes; consequently, the cost model accounts for both an operation's complexity and its effect on data irregularities.

For the complexity estimation, each evolution operation is formalized in an implementation near manner, as exemplified in Table 2 for `add`. This allows users to predict the complexity of the Cypher code. In alignment with our evolution language GEO, every operation is partitioned by entity types (nodes and edge) and features (label, property). In our notation, O_a represents the aligning Cypher command, in this context CREATE or MERGE. L is the finite set of available labels, subdivided into node labels L_N and edge labels L_E, whereby l_n and l_e resemble a precise instance. Properties are defined as key-value pairs $K \times V$, whereby n is a Cypher variable to refer to a defined pattern, like in `CREATE(n:Client)`. Since GEO translates directly into Graph Manipulation Operations (GMOs), the formalization is based on the aligning GEO statement.

To ensure a cost-aware schema evolution and schema transformation, the cost function are further refined by the following aspects derived from the metadata stored in the GDPs and categorized according to [20].

- *Structure metadata*: The number of affected entities to estimate pattern complexity.
- *Cardinalities and value distribution*: Analysis of whether constraints become invalid as heterogeneity increases or whether new constraints emerge by homogenizing former irregularities.
- *Paths metadata*: Identification of important nodes with numerous edges, prone to cause a structural break upon their modification.
- *Patterns, data types & domains*: Use of syntactic distance and semantic similarity to detect potential heterogeniouse structures.
- *Functional*: Deriving graph inclusion dependencies to detect renamed, copied or moved structures in and between versions or databases.

For each aspect, a weight is defined in association of its effect on heterogeneous structures, whereby a value of one resembles homogeneity. Costs are sensitive to irregularities, thus, requiring the composition of GDP metadata and operation complexity. Our approach considers cost-estimations for monitoring data engineering pipelines and supports user decisions via a GDP Diff preview together with the emerging target schema in terms of schema evolution. The tool displays the resulting costs approximation to the user.

6. Data Quality Aspects

The framework considers two data quality dimensions:

Syntactic Accuracy The integration of GDPs as well as the GDP Diff preview aim to support data cleansing by improving syntactic accuracy. This is accomplished by depicting candidates for typical errors occurring in relaxed schemata, such as typos, by measuring the syntactic distance. Subsequently, users can utilize the evolution form to clean the data accordingly (see *Use Case 1*). Since users cannot estimate the impact of an evolution operation on the dataset, we propose a GDP Diff preview serving as control mechanism before actually conducting the operation on the original dataset. Thus, the GDPs depict heterogeneity during data exploration caused by the schema-less environment. In addition, the GDP Diff allows estimating the impact of schema evolution and gives insight into irregularity changes during schema transformation (see *Use Case 2*).

Semantic Consistency The GDPs specialize in detecting semantic errors to facilitate user-driven decision making, thereby, increasing data consistency. To achieve this, the semantic similarity between entity types and properties is extracted, allowing refinements of structures that emerged unintentionally during schema evolution. On top of that, constraints are included to illustrate homogeneous parts suitable to be put under schema control. Regarding data dependencies, GDPs store information on valid inclusion dependencies, which provide insights into (1) identical in-version information as well as (2) evolution or transformation operations like `rename`, `move` or `copy`. Each GDP represents the left side of an inclusion dependency. Moreover, the GDP Diffs make increases and decreases in semantic irregularities during monitoring tasks visible.

7. Implementation

Nautilus implements our evolution language Geo, closing the gap for an evolution language for property graphs. Geo aims to widen the range of users by intuitively defining each operation, requiring no prior knowledge of Cypher. The implementation is available on GitHub[1]. The tool includes a form to define the renown evolution operations `add`, `rename`, `delete`, `copy`, `move`, `spit`, `merge` and the graph-specific `transform` operation (i.e., converting a node into an edge and vice versa). Moreover, a first version of the GDPs is integrated in a data exploration option. This includes information on label occurrences; partitioned by nodes and edges. For each label all associated properties of the initial and latest schema versions are depicted, allowing a manual comparison.

We plan to extend the current data profiles in Nautilus by integrating the GDPs. This will highlight irregularities in the source graph and enable users to clean variations by evolving the schema with our evolution form. After defining the evolution operation, the resulting GDP Diff (see Table 2) is displayed in a preview along with the estimated costs. This supports users in estimating the impact of an operation on the schema and its underlying data before executing it on the original data. To evaluate the accuracy of the presented GDPs, we will generate a dataset with heterogeneous structures, such as optional properties, missing edges, and variations, serving as gold standard. Initial tests revealed a high accuracy of the current data profile generation. In the future, we will extend the evaluation by determining precision, recall and F1 score derived from true positives, false positives, and false negatives.

[1]Nautilus: Implementation of a graph evolution language: https://github.com/DominiqueHausler/Nautilus-Graph-Schema-Evolution

Operation		Formalized Graph Manipulation Operation	Evolution language – Geo
add	node	$O_a(n : l_n \{P\})$ with $O_a \in \{\text{CREATE, MERGE}\}$, $P \subseteq K \times V, l_n \in L_N$	"add node with label" **label**
	edge	$O_a(n_i) - [n : l_e \{P\}] \rightarrow (n_j)$ with $O_a \in \{\text{CREATE, MERGE}\}$, $P \subseteq K \times V, l_e \in L_E$	"add relationship with type" **label**
	label	SET $n : l_n$ outputting $L'_N(n) = L(n) \cup \{l_n\}$	"add label" **label** "to node with label" **labels**
	property	SET $n.p = v$ with $P'(n) = P(n) \cup \{(k, v)\}$	"add" ("unique")? ("mandatory")? "property" **property** "with datatype" **datatype** "to node with label" **label**

Table 2
Formalization of `add` for the cost functions, aiming to illustrate operation complexity. These are based on Geo from which Cypher code can be derived. The `add` operation is partitioned by node, edge, label and property.

8. Conclusion

Our presented framework facilitates user-informed decision making in the terms of schema evolution and transformation – two closely intertwined approaches. For schema transformation, the framework outlines heterogeneity changes in form of Graph Data Profile Diffs (GDP Diffs), additionally calculating the costs for each transformation operation. Thus, the framework can be seamlessly integrated into monitoring components of data engineering pipelines. Evolutionary tasks benefit from a GDP Diff preview, allowing users to estimate an operation's impact and costs before executing it on the original data. Additionally, GDPs highlight potential errors through heterogeneity metrics, enabling the user to improve data quality by defining evolution operations to clean them. To determine whether heterogeneity in or between versions increased, decreased or remained consistent, the cost functions incorporate the information stored in the GDPs alongside the formalized operation. Users, therefore, are provided a precise estimation of an operation's impact together with a comprehensive analysis of relaxed schemata in schema-less graph databases.

Acknowledgments

The work of Dominique Hausler has been funded by Deutsche Forschungsgemeinschaft (German Research Foundation) – 552691270.

Declaration on Generative AI

The logo was generated with DALL-E 3. ChatGPT and AI Studio were used for rephrasing and to find synonyms.

References

[1] Z. Brahmia, F. Grandi, B. Oliboni, A literature review on schema evolution in databases, Computing Open 02 (2024) 2430001.

[2] T. Eckwert, M. Guckert, G. Taentzer, EvolveDB: Evolving relational database schemas in a model-driven way, Software and Systems Modeling (2025) 1–26.

[3] A. Cleve, M. Gobert, L. Meurice, J. Maes, J. H. Weber, Understanding database schema evolution: A case study, Sci. Comput. Program. 97 (2015) 113–121.

[4] U. Störl, M. Klettke, S. Scherzinger, NoSQL schema evolution and data migration: State-of-the-art and opportunities, in: EDBT, OpenProceedings.org, 2020, pp. 655–658.

[5] A. Bonifati, P. Furniss, A. Green, R. Harmer, E. Oshurko, H. Voigt, Schema validation and evolution for graph databases, in: ER, volume 11788 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 448–456.

[6] R. Angles, A. Bonifati, S. Dumbrava, G. Fletcher, A. Green, J. Hidders, B. Li, L. Libkin, V. Marsault, W. Martens, F. Murlak, S. Plantikow, O. Savkovic, M. Schmidt, J. Sequeda, S. Staworko, D. Tomaszuk, H. Voigt, D. Vrgoc, M. Wu, D. Zivkovic, PG-Schema: Schemas for Property Graphs, Proc. ACM Manag. Data 1 (2023) 198:1–198:25.

[7] D. Tomaszuk, J. E. L. Gayo, On Property Constraints in PG-Schema, in: K-CAP, ACM, 2025, pp. 69–73.

[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. Savkovic, M. Schmidt, J. F. Sequeda, S. Staworko, D. Tomaszuk, PG-Keys: keys for Property Graphs, in: SIGMOD Conference, ACM, 2021, pp. 2423–2436.

[9] A. Bonifati, S. Dumbrava, E. Martinez, F. Ghasemi, M. Jaffré, P. Luton, T. Pickles, DiscoPG: Property Graph schema discovery and exploration, Proc. VLDB Endow. 15 (2022) 3654–3657.

[10] S. Sideri, G. Troullinou, E. Ymeralli, V. Efthymiou, D. Plexousakis, H. Kondylakis, PG-HIVE: hybrid incremental schema discovery for Property Graphs, CoRR abs/2512.01092 (2025).
[11] J. F. Terwilliger, P. A. Bernstein, A. Unnithan, Worry-free database upgrades: automated model-driven evolution of schemas and complex mappings, in: SIGMOD Conference, ACM, 2010, pp. 1191–1194.
[12] E. Rahm, P. A. Bernstein, A Survey of Approaches to Automatic Schema Matching, VLDB J. 10 (2001) 334–350.
[13] G. Ding, S. Sun, G. Wang, Schema matching based on SQL statements, Distributed Parallel Databases 38 (2020) 193–226.
[14] W. Martens, M. Niewerth, T. Popp, C. Rojas, S. Vansummeren, D. Vrgoc, Representing paths in graph database pattern matching, Proc. VLDB Endow. 16 (2023) 1790–1803.
[15] P. Barceló, J. Pérez, J. L. Reutter, Schema mappings and data exchange for graph databases, in: ICDT, ACM, 2013, pp. 189–200.
[16] N. Francis, L. Libkin, Schema mappings for data graphs, in: PODS, ACM, 2017, pp. 389–401.
[17] J. Bruyat, P. Champin, L. Médini, F. Laforest, PRSC: from PG to RDF and back, using schemas, Semantic Web 15 (2024) 2555–2595.
[18] V. Rafe, M. Golparian, S. Rasoolzadeh, Using graph transformation systems to formalize tropos diagrams, J. Vis. Lang. Comput. 30 (2015) 1–16.
[19] J. L. Andersen, A. Davoodi, R. Fagerberg, C. Flamm, W. Fontana, J. Kolcák, C. V. F. P. Laurent, D. Merkle, N. Nøjgaard, Automated inference of graph transformation rules, CoRR abs/2404.02692 (2024).
[20] S. Maiolo, L. Etcheverry, A. Marotta, Data profiling in Property Graph databases, ACM J. Data Inf. Qual. 12 (2020) 20:1–20:27.
[21] L. C. Shimomura, G. H. L. Fletcher, N. Yakovets, ProGGD - data profiling on Knowledge Graphs using graph generating dependencies, in: ISWC (Posters/Demos/Industry), volume 3632 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023.
[22] M. Manouvrier, K. Belhajjame, Graph functional dependencies: Analysis and translation to PG-Schema, Inf. Syst. 136 (2026) 102633.
[23] D. Hausler, A. Conrad, M. Sperling, U. Störl, M. Klettke, Discovering inclusion dependencies in a multi-model scenario, in: ER, volume 16189 of *Lecture Notes in Computer Science*, Springer, 2025, pp. 242–260.
[24] M. Klettke, H. Awolin, U. Störl, D. Müller, S. Scherzinger, Uncovering the evolution history of data lakes, in: IEEE BigData, IEEE Computer Society, 2017, pp. 2462–2471.
[25] R. de Oliveira Werneck, R. Raveaux, S. Tabbone, R. da Silva Torres, Learning cost functions for graph matching, in: S+SSPR, volume 11004 of *Lecture Notes in Computer Science*, Springer, 2018, pp. 345–354.
[26] J. AbuBekr, I. Chikalov, S. Hussain, M. Moshkov, Sequential optimization of paths in directed graphs relative to different cost functions, in: ICCS, volume 4 of *Procedia Computer Science*, Elsevier, 2011, pp. 1272–1277.
[27] C. Chen, G. Hu, D. Zuo, C. Yu, Y. Ma, H. Zhang, E-Syn: E-graph rewriting with technology-aware cost functions for logic synthesis, in: DAC, ACM, 2024, pp. 124:1–124:6.
[28] M. Neuhaus, H. Bunke, Automatic learning of cost functions for graph edit distance, Inf. Sci. 177 (2007) 239–247.
[29] S. Marchesin, G. Silvello, O. Alonso, Large language models and data quality for Knowledge Graphs, Inf. Process. Manag. 62 (2025) 104281.
[30] P. Sadhukhan, S. Gupta, A graph theoretic approach to assess quality of data for classification task, Data Knowl. Eng. 158 (2025) 102421.
[31] F. Sidi, P. H. S. Panah, L. S. Affendey, M. A. Jabar, H. Ibrahim, A. Mustapha, Data quality: A survey of data quality dimensions, in: CAMP, IEEE, 2012, pp. 300–304.
[32] L. Ehrlinger, W. Wöß, A survey of data quality measurement and monitoring tools, Frontiers Big Data 5 (2022) 850611.
[33] D. Hausler, M. Klettke, U. Störl, A language for graph database evolution and its implementation

in Neo4j, in: ER (Companion), volume 3618 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023.
[34] D. Hausler, M. Klettke, Nautilus: Implementation of an evolution approach for graph databases, in: MoDELS (Companion), ACM, 2024, pp. 11–15.