GRIP: Constraint-based Explanation of Missing Answers for Graph Queries

Qi Song
Amazon.com
qison@amazon.com

Hanchao Ma
Case Western Reserve University
hxm382@case.edu

Peng Lin
Washington State University
peng.lin@wsu.edu

Yinghui Wu
Case Western Reserve University,
Pacific Northwest National Laboratory
yxw1650@case.edu

ABSTRACT

A useful feature in graph query engines is to clarify "Why certain entities (nodes, attribute values or edges) are missing" in query answers. This task is even more challenging when the relevant data is already missing in the underlying data source. Missing data, on the other hand, can be inferred by enforcing data constraints for graphs. We demonstrate GRIP, a system that exploits data constraints to clarify missing answers for graph queries. (1) *Constraint-based explanation.* Given a desired yet missing entity in the query answer, GRIP ensures to generate finite and minimal sequences of data constraints (an "explanation") that should be consecutively enforced to G to ensure its occurrence for the same query. (2) *Answering "why" and "how" questions.* Users can query GRIP with both "Why" ("Why" the element is missing) and "How" questions ("How" to refine the graph to include the missing answer). GRIP engine supports runtime generation of explanations by incrementally maintaining a set of bi-directional search trees. (3) *Interactive exploration.* GRIP provides user-friendly GUI to support interactive ad visual exploration of explanations, including both automated generation and step-by-step inspection of graph manipulations.

CCS CONCEPTS

• **Information systems** → **Data provenance**; **Database query processing**.

KEYWORDS

Graphs, Data Constraints, Data Provenance

ACM Reference Format:
Qi Song, Hanchao Ma, Peng Lin, and Yinghui Wu. 2021. GRIP: Constraint-based Explanation of Missing Answers for Graph Queries. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21), June 20–25, 2021, Virtual Event , China.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3448016.3452758

SIGMOD '21, June 20–25, 2021, Virtual Event , China

ACM ISBN 978-1-4503-8343-1/21/06…$15.00
https://doi.org/10.1145/3448016.3452758

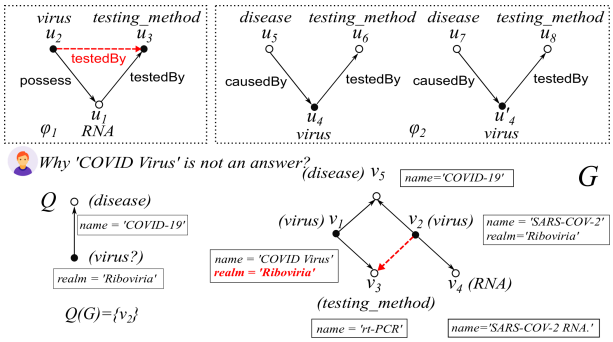

Figure 1: Explain missing answers in medical search.

1 INTRODUCTION

A useful feature of graph query systems is to clarify why a specific entity (*e.g.,* a node, an attribute value of a node, or an edge) is missing in the query answer. Given the query answer $Q(G)$ of a query Q (*e.g.,* SPARQL) in an (incomplete) graph G, and a desired yet missing entity g that should be in $Q(G)$, the problem is to identify the (missing) fraction of G, denoted as ΔG, such that the entity g occurs in the query answer $Q(G')$. Why-provenance [1] clarifies missing answers with the existing fraction of data source that is responsible for missing answers. The problem for graph data is nevertheless more involved, especially when the data that is responsible for the missing answer may also be missing.

The missing data, on the other hand, can be inferred by enforcing data constraints [3–6, 9]. Data constraints for graphs identify node pairs (v, v') in G via graph pattern matching between a graph pattern P and G, and either enforce node equivalence or assert a missing edge between v and v', for any pair (v, v') from a same matching function (simply "matching", typically defined by subgraph isomorphism). For example,

- Graphs keys [3] states that "the two nodes that match the designated pattern nodes in the same matching are equivalent and should refer to the same real-world entity";
- Graph association rules [4, 6, 8] state that "there is an edge between two nodes if they are induced from a same matching between P and G".

Can data constraints be used to explain "why" specific entity of interests is missing in graph search? Naturally, the enforcement of a data constraint may introduce new nodes and edges, hence provides useful provenance information towards recovering missing query

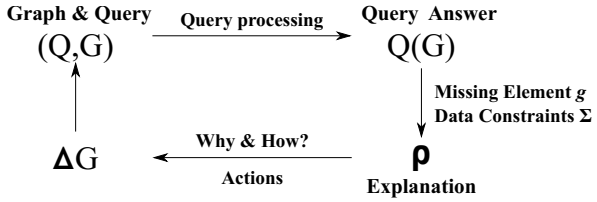

Figure 2: Workflow of GRIP.

answers. Moreover, this helps us understand how to "manipulate" the graph towards desired answers.

Example 1: Consider a medical study scenario over a knowledge base about COVID-19 illustrated in Fig. 1 (excluding the fraction marked in red). A user starts with a SPARQL query Q "*find all viruses that may be relevant to COVID-19 and has a realm 'Riboviria'*" While the answer $Q(G)$ contains a relevant virus "SARS-COV-2" (v_2), another entity v_1 'COVID Virus' that is known to be relevant is not in the answer. Using G alone does not clarify whether v_1 indeed belongs to the missing answer of Q, or is simply irrelevant.

Two data constraints are validated for the facts in G.

> φ_1 (Graph association rule): "if a virus possesses a specific RNA which can be identified by a testing method, then this method can be used to identify the virus".

> φ_2 (graph key): "if two viruses cause the same disease and can be identified by the same testing method, then they refer to the same realm".

Enforcing φ_1 "inserts" a missing edge between v_2 and v_3. A follow-up enforcement of φ_2 enriches the missing "realm" information of virus v_1. This enriches G and restores v_1 in $Q(G)$. An explanation can then be characterized by a sequence of enforcement of φ_1 and φ_2, along with useful provenance information such as the "manipulations" and corresponding data changes of G. □

Several methods have been developed to complete knowledge graphs [2, 7]. These methods aim to infer new data rather than relevant information for clarifying missing answers. Moreover, it is desirable to compute a minimal amount of manipulation to include missing answer, rather than completing the entire graph.

GRIP. These motivate us to develop GRIP, a system that can interactively search and explain missing query answers with constraints [9]. It has the following unique features.

Constraint-based explanation. GRIP is designed to infer at query-time only a necessary amount of data that is responsible for user-specified missing answers. It can clarify missing nodes, attribute values, or edges. GRIP characterize explanations as non-destructive "actions" that only enrich graphs with new information. Each action specifies a data constraint, a node pair to be operated on, and a graph editing operator. By exploiting validated data constraints (*e.g.,* graph keys and graph association rules), GRIP automatically computes a minimal sequence of actions that is responsible for the occurrence of a missing element. It guarantees finite and minimal explanations, and efficiently computes the explanations with a bi-directional inference algorithm.

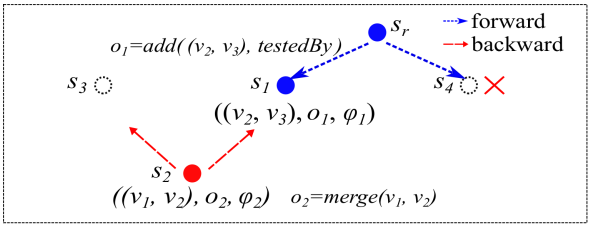

Figure 3: Bidirectional search for explaining missing answer v_1.

Answering "Why" and "How" questions. GRIP supports efficient querying of the explanations. It incrementally maintains the provenance information as cost-effective provenance trees. Users can easily request explanations by asking ad-hoc "Why" and "How" questions, which asks why a specific element is missing (to find relevant data constraints) and how to manipulate graphs to include the missing entities for the query (to inspect the useful actions and suggested changes to the graph data).

Interactive exploration. GRIP browsers provide user-friendly GUI for users to visually explore the explanation and provenance trees (with step-by-step exploration), inspect changes and responsible data constraints, and track the incremental maintenance of the explanations upon new requests. We also demonstrate the application of GRIP in validating data constraint quality and graph refinement.

Below we overview GRIP (Section 2) and its major workflow, key enabling techniques and system architecture. We demonstrate its ease of use and effectiveness, and show its applications (Section 3).

2 SYSTEM OVERVIEW

2.1 Workflow of GRIP

The interactive workflow in GRIP consists of multiple search sessions. A single session is illustrated in Fig. 2). (1) It starts by processing a graph query Q over (a possibly incomplete) graph G and returns the query answer $Q(G)$ to users for inspection. (2) Given $Q(G)$, a set of data constraints Σ, and a user specified missing query answer g (which can be a node, a node attribute, or an edge) that may not be even in G, GRIP either generates a minimal explanation ρ as a sequence of actions or returns $\emptyset$ ("not explainable"; to be discussed). (3) Users may then specify ad-hoc "why" and "how" questions, on any inferred new data. GRIP returns specific manipulations of graph ΔG accordingly for users to validate. (4) Upon validated manipulation, GRIP commits the updates to G and resumes to step (3), or triggers the next session upon a new query.

We next introduce the key enabling techniques of GRIP.

2.2 Constraint-based Explanation

Data constraints. A graph data constraint φ has a general form $\mathcal{P} \rightarrow X$, where $\mathcal{P}$ is a graph pattern and X is a value constraint [9]. GRIP supports two classes of constraints that:

- enforce node equality (NE): $\mathcal{P} \rightarrow u_o.\text{id} = u'_o.\text{id}$; or
- capture missing edges (EG): $\mathcal{P} \rightarrow \exists r(u_o, u'_o)$

Graph patterns. The pattern $\mathcal{P}$=($P(u_o, u'_o)$,L) consists of a graph pattern $P(u_o, u'_o)$ with labeled pattern nodes and edges, and a set of value constraints L defined on node attributes of P. There are two designated "entity nodes" u_o and u'_o in P. A *matching* between $\mathcal{P}$ and a graph G is a subgraph isomorphism that (1) maps each

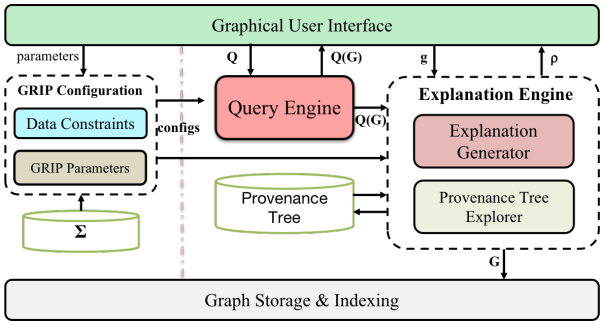

Figure 4: Architecture of GRIP

pattern node (resp. edge) in $P(u_o, u_o')$ to a node (resp. edge) with a same label (a match) in G, and (2) ensures all matches satisfy the constraints L. A node pair (v, v') is a match of $\mathcal{P}$ (and of a NE or EG $\mathcal{P} \rightarrow X$), if v and v' matches u_o and u_o' in a same matching.

Semantics. A NE (resp. EG) φ with pattern $\mathcal{P}$=$(P(u_o, u_o'), L)$ states that "*for any match* (v, v') *of* (u_o, u_o'), v *and* v' *are equivalent and should refer to a same entity*" (resp. "*has a missing edge* $r(v, v')$"). Given a graph G=(V, E) with nodes V and edges E, a match (v, v') of a NE (resp. EG) φ is a *violation* of φ if v.id $\neq v'$.id (resp. $r(v, v') \notin E$).

Actions and sequences. Given a graph G and constraints Σ, an action s is a triple $((v, v'), o(v, v'), \varphi)$, where (v, v') is a violation of φ in G, and $o(v, v')$ is an operator that either "merges" v and v' and enriches their missing attribute values (given the node equivalence), or inserts a missing edge $r(v, v')$, thus removes the violation by enforcing φ on (v, v'). The result of s on G, denoted as G^s, refers to the graph obtained by applying $o(v, v')$ on G.

The result of a sequence $\rho = \{s_1, ..., s_n\}$ of actions from G is a graph G^ρ obtained by sequentially applying the operators in the actions. A sequence is terminating if no action can extend it towards a new graph ($G^{\rho \cdot a} = G^\rho$ for any action a).

Σ-explainable. Given the query answer $Q(G)$ of a query Q in graph G, and constraints Σ, a missing answer g not in G is Σ-*explainable* if there is a non-empty sequence ρ such that g is in $Q(G^\rho)$. We say ρ is an explanation of g *w.r.t.* Σ,Q and G [9]. GRIP process is justified by a *Church-Rosser* property [9].

Computing Minimal Explanations. GRIP explains missing answers with informative and minimal explanations. A minimal explanation is a sequence where any sub-sequence is not an explanation. Furthermore, it solves the following problem:

$$\rho = \arg\max_{|\rho'| \leq b} \mathrm{cg}(\rho', G)$$

where $\mathrm{cg}(\rho', G)$ quantifies the cumulative informativeness gain (the amount of new information) by applying the operators in ρ' [9], and b is a tunable size bound of explanations (set as 3 by default).

Bi-directional algorithm. GRIP performs a bidirectional Breadth-First search (up to size bound b) over a partially observed tree which contains possible actions. The forward search starts from G with a root action s_r and explores admissible actions, while the backward search starts with a "virtual" action s_g that contains g and "reverse engineers" Σ enforcement to explore a set of enabling actions that may result in this action. The bidirectional search stops until a common action s is identified. An explanation is constructed as the sequence from s_r to s_g passing s. As there may exist multiple

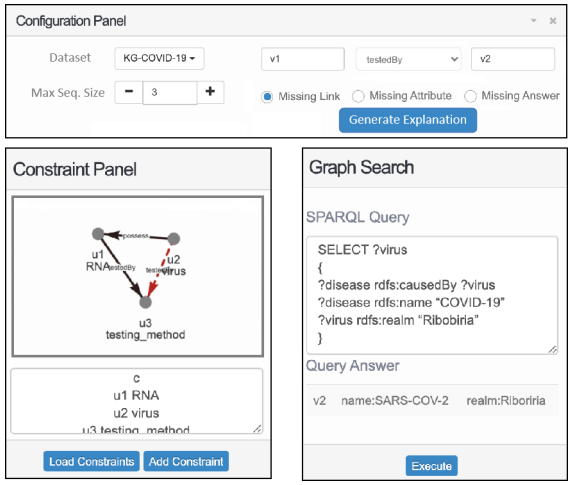

Figure 5: GRIP User Interface: Configuration

s in the intersection, the best explanation that maximizes accumulated gain cg is then returned. During the bi-directional search, GRIP keeps tracking if the action infers the missing data to recover g as a part of the query answer.

GRIP ensures to generate finite, minimal and optimal (most informative) explanations. The overall explanation cost is in $O(T \cdot (|V|^2|\Sigma|^{b/2}))$. Here T is the time cost for detecting new missing data triggered by an action.

Example 2: Fig. 3 illustrates the bidirectional search to explain a missing answer in Example 1. (1) It initializes a forward tree from a root action s_r and explores possible actions s_1 and s_4 (details of constraints and violations are omitted). (2) It constructs a "virtual" action s_2 that contains g and explores actions that may result in this action, *i.e.,* s_1 and s_3. (3) The bidirectional search stops at s_1 since it is a common action for both forward and backward search. GRIP then returns an explanation $\{s_1, s_2\}$ to users. □

Exploring explanations. GRIP tracks bi-directional search by maintaining a class of provenance trees, where each node is an action, and there is an edge (s, s') between two actions if s and s' are explored consecutively in a sequence. It helps users to query the provenance information with three types of questions. (1) "***Why*** *a specific entity* g *is missing in* $Q(G)$*?*" To answer this question, GRIP retrieves the optimal explanation ρ of g from the provenance tree that leads to the inclusion of g in $Q(G^\rho)$, along with involved constraints and violations. (2) "***How*** *to manipulate* G *to recover* g*?*" GRIP extracts the corresponding sequence of the operators from ρ ("merge" and "insert") and the corresponding changes posed to the graph G. (3) While provenance trees only store the explored fraction that is relevant to clarify g, GRIP also helps users with ad-hoc "**What-if**" analysis ("What can be inferred if I start from a specific action of interests?") GRIP guides the user to explore, in a step-by-step mode, (a) what new data can be inferred and (b) the responsible constraints and actions.

2.3 GRIP Architecture

GRIP adopts a three-tier architecture depicted in Fig. 4. (1) The top layer is an interactive interface that allows users to visually configure, and input missing elements. We demonstrate the user-friendly

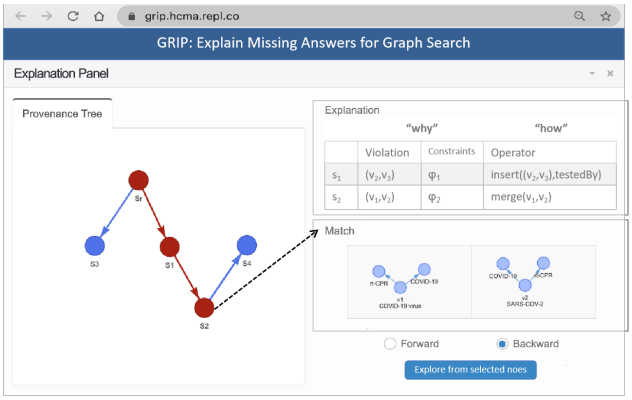

Figure 6: GRIP **Explanation of missing answer**

GUI in Section 3. (2) The configuration component allows users to load or input data constraints and input required parameters. (3) The query engine processes graph queries *e.g.*, SPARQL. (4) The explanation engine generates explanations and allow users to explore the provenance tree. It interacts with the query engine to explain missing query answers. (5) The storage and index layer provides fast access to attributed graphs.

3 DEMONSTRATION

Setup. We demonstrate GRIP with real-world graphs and show its application in understanding missing informations: (1) KG_COVID[1], an knowledge graph of 15M entities (*e.g.*, virus, disease, RNA) and 38M edges. (2) DBYa [9], with 592K nodes, 4.5M edges, and 50K equivalent pairs with aligned attributes curated from knowledge bases DBPedia and YAGO; (3) DBIM [9], which contains 33K nodes, 200K edges and 33.4K entities covering 10 types of equivalent pairs across DBPedia and IMDb (a movie knowledge base); (4) OAG [9], an open academic graph which unifies Microsoft Academic Graph and Aminer with 2.5M nodes, 5.2M edges. We use Apache Jena[2] as the SPARQL query engine. We also provide a web portal[3] and the source code of GRIP[4].

Scenarios. We showcase the following scenarios.

Explaining missing answers for SPARQL. We invite the users to interact with the user-friendly GUI of GRIP to search graphs with SPARQL queries and easily track explanations for missing answers (Fig. 6). Accessing the "Configuration" panel at any session, users are able to select real-world graphs and tune the size of explanation. Users can construct SPARQL queries using "Graph Search" panel. The constraints are visualized in the "Constraint" panel. The provenance tree is visualized in the "Explanation" panel. Users can inspect the rich information in the tree nodes of provenance trees, such as the constraints and their violations, the inferred missing data and responsible operators. We show that GRIP is capable to explain missing query answers with data that is not in the current dataset with graph data constraints. When no query is give, GRIP can be used to explain why a certain attribute value of a node or a certain edge is missing.

[1]https://github.com/Knowledge-Graph-Hub/kg-covid-19
[2]https://jena.apache.org/
[3]https://grip.hcma.repl.co/
[4]https://github.com/wsu-db/GRIP/

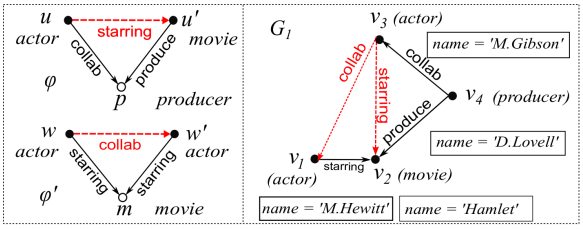

Figure 7: Clarifying inaccurate elements.

Queryable Explanations. We invite users to query and explore the provenance information with "Why", "How" and "What-if" questions. A user can select a node in the provenance tree to pose any of these questions. GRIP responses by highlighting the relevant constraints, violations, actions and operations. Figure 6 illustrates the answer of "why the 'realm' information of node v_1 is missing?", where the sequence $\rho = \{s_1, s_2\}$ is highlighted with involved violations, attributes values and constraits.

Interactive exploration. In this scenario, we invite users to interactively browse and explore the provenance tree, and inspect how the provenance tree are generated in a controllable way. In each session, a user can select any node in the provenance tree and inspect a step-by-step forward/backward exploration rooted at the selected node. Following the similar forward/backward search used by Σ-explanation, GRIP incrementally extends the provenance tree and infers more missing data as requested. Figure 5 demonstrates a forward exploration from provenance node s_2, which triggers an action s_5 to remove a new violation of φ_3 (not shown).

Applications. We show that GRIP can be used in real-world applications. (1) If no query and missing entity is specified, GRIP performs budgeted graph refinement by inferring new elements with Σ. (2) GRIP can be used to validate if any constraint is responsible for inferring "erroneous" entities. Fig. 7 illustrates a fragment of DBIM (G_1) with two graph association rules:

> (φ): *"an **actor** (u) **stars** in a **movie** (u') if a **producer** (p) he collaborates also produces the same movie".*

> (φ'): *"an **actor** (w) **collaborates** with an **actor** (w') if they both starred a movie (m)."*

The insertion of an edge collab (v_1, v_3) is annotated as "inaccurate" (*'M.Gibson' and 'M.Hewitt' collaborated in a movie*). BiExp generates an explanation, which states that an edge insertion $\oplus((v_3, v_2), \text{starring})$ (by enforcing φ) leads to the inaccurate element collab (v_1, v_3) (by enforcing φ'). A closer inspection suggests that φ can be an "overkill", given the exception of (v_3, v_2) ("an actor may not always be starring a movie produced by a producer he collaborated with").

Performance. GRIP tracks the performance such as session response time, and visualizes analytics of factors such as number of constraints and cost bound. GRIP is able to generate explanations for a missing query efficiently, for example, it takes on average 5 seconds to generate an explanation over KG_COVID with 50 constraints.

Acknowledgement This work is supported by NSF under CNS-1932574, OIA-1937143, ECCS-1933279, CNS-2028748, DoE under DE-IA0000025, USDA under 2018-67007-28797, and PNNL Data-Model Convergence initiative.

REFERENCES

[1] James Cheney, Amal Ahmed, and Umut A Acar. 2007. Provenance as dependency analysis. In *DBPL*. 138–152.

[2] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. 2014. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *KDD*.

[3] Wenfei Fan, Zhe Fan, Chao Tian, and Xin Luna Dong. 2015. Keys for graphs. *PVLDB* 8, 12 (2015).

[4] Wenfei Fan, Xin Wang, Yinghui Wu, and Jingbo Xu. 2015. Association rules with graph patterns. *PVLDB* (2015).

[5] Wenfei Fan, Yinghui Wu, and Jingbo Xu. 2016. Functional dependencies for graphs. In *SIGMOD*.

[6] Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian Suchanek. 2013. AMIE: association rule mining under incomplete evidence in ontological knowledge bases. In *WWW*.

[7] Ryan N Lichtenwalter and Nitesh V Chawla. 2011. Lpmade: Link prediction made easy. *JMLR* (2011).

[8] Peng Lin, Qi Song, Jialiang Shen, and Yinghui Wu. 2018. Discovering Graph Patterns for Fact Checking in Knowledge Graphs. In *DASFAA*.

[9] Hanchao Ma Yinghui Wu Qi Song, Peng Lin. 2021. Explaining Missing Data in Graphs: A Constraint-based Approach.