

# Grounding Provenance for Graph Inference with Data Constraints

Haolai Che, Mingjian Lu, Yangxin Fan, Hanchao Ma, Yinghui Wu

Case Western Reserve University

Cleveland, Ohio, USA

{hxc859,mxl1171,yxf451,hxm382,yxw1650}@case.edu

## ABSTRACT

Graph machine learning models have been routinely queried (“graph inference”) in various applications. To help users clarify and understand their output, provenance and explanation methods are studied to generate subgraphs to explain model outputs. Nevertheless, they may be meaningless and misleading due to misalignment with real-world evidences. This paper introduces a novel data provenance approach to generate subgraphs with high interpretability and meanwhile grounded by genuine evidence from data constraints. (1) We formulate a class of *grounded witnesses*, which can suggest critical subgraphs for the output of graph inference in terms of counterfactual analysis, and grounded evidence from user-defined data constraints. (2) We propose quality measures for grounded witnesses, and formulate the problem of witness generation under constraints. We establish the hardness results for the problem. (3) We provide an algorithmic framework, based on a variant of Chase-and-Backchase algorithm with bounded rounds of reasoning and graph inference verification. We specify the general framework with two efficient algorithms. The first algorithm selectively “reverse” and enforce data constraints to make witness graphs grounded by ensuring the co-occurrence of antecedents and consequents of the constraints. The second efficiently extracts arborescences that satisfy the data constraints. Using both real-world and synthetic benchmark datasets, we verify that our algorithms efficiently computes well-grounded provenance structure and scale well to large graphs and query load. We also showcase on their application in real-world graph analytical tasks.

## PVLDB Reference Format:

Haolai Che, Mingjian Lu, Yangxin Fan, Hanchao Ma, Yinghui Wu.

Grounding Provenance for Graph Inference

with Data Constraints. PVLDB, 14(1): XXX-XXX, 2020.

doi:XX.XX/XXX.XX

## PVLDB Artifact Availability:

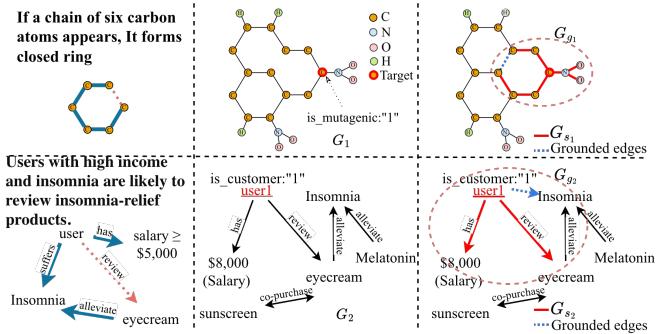
The source code, data, and/or other artifacts have been made available at [https://github.com/anthonyche/Grounded\\_Witness](https://github.com/anthonyche/Grounded_Witness).

## 1 INTRODUCTION

Graph machine learning (ML) models, such as graph neural networks (GNNs), have shown promise results in graph analytical

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit <https://creativecommons.org/licenses/by-nc-nd/4.0/> to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.  
doi:XX.XX/XXX.XX



**Figure 1: Grounding graph inference analysis with genuine evidence declared in data constraints.**

applications. A graph model  $M$  can be considered as a function that takes as input a representation of a graph  $G = (X, A)$ , where  $X$  (resp.  $A$ ) is a matrix of node features (resp. adjacency matrix) of  $G$ , and transforms  $G$  to a (numerical) matrix  $Z$  (“logits”) via an inference process. The logits  $Z$  can then be post-processed to task-specific output for downstream analysis, such as labels for node classification. Given a test graph  $G$  and a trained GNN model  $M$ , an “inference query”  $Q(M, G)$  applies the inference process of  $M$  over  $G$  to return the desired, user-specific output (set as  $Z$  by default). In practice, an inference query can also designate a set of test nodes  $V_T$  to only return results of interest in the form of  $Q(M, V_T, G)$ .

While promising, secured, trustworthy application of GNNs requires reliable provenance and interpretation of GNNs’ inference and outputs. In an analogy to Why-provenance [5], such need can be expressed by a Why-question posed on an inference query that treats a GNN as an “oracle”, and computes subgraphs (“witnesses”) that can clarify “Why” a result exists in the inference output. On the other hand, real-world graphs are often *incomplete*. Erroneous and incomplete data may be carried over by a straightforward provenance analysis, yielding inconsistent, unconvincing provenance results. Consider the following example.

**Example 1:** We illustrate two scenarios as shown in Fig. 1.

(1) **[Molecular Property Analysis].** Chemists heavily rely on empirical rules (facts) or requirements of occurrences of chemically valid structures for molecular analysis, e.g., carbon-rings. Such a rule can be specified by a data constraint  $\varphi_1$ : “*If a chain of six carbon atoms appears ( $P_1$ ), it forms a closed ring ( $c_1$ )*”.  $G_1$  illustrates a molecule labeled as mutagenic, where the red node is classified by a 3-layer GCN  $M$  as a *mutagenic atom*-i.e., an atom that contributes to the molecule’s mutagenicity. There are missing edges in  $G_1$ : two benzene-like structures should have been formed as a stable two 6-atom rings; instead, they are presented as a 10-atom ring. A direct

explanation, by e.g., a GNN explainer or a provenance analysis, leads to a misleading interpretation (marked by red as  $G_{s_1}$ ), which has little chemical meaning and may not be valid in the real world.

(2) [Web recommendation]. Consider another scenario that arises in social recommendation systems. A typical customer behavior pattern  $\varphi_2$ : “*If a user has a salary above \$5000 and suffers from insomnia ( $P_2$ ), he/she is likely to review or purchase an eye cream that alleviates insomnia’s side effects ( $c_2$ )*”. In the purchasing graph  $G_2$ , a recommender GNN predicts that user1 (in red) is a potential buyer of a specific type of eye cream.

A business analyst may ask: “*Why is this user classified as a potential buyer?*” Ideally, the provenance should link this decision to the user’s insomnia condition and the product’s therapeutic relation (*alleviates insomnia*). Due to missing information indicating the user’s insomnia, the provenance relies solely on historical correlations between eye cream and income level, while ignoring the health condition that actually drives the purchase. The analyst may misinterpret such an interpretation and suggest unrelated products (e.g., luxury “sunscreen”), yielding irrelevant recommendations that deviate from the true purchase intention.  $\square$

The above cases highlight the need to make graph inference grounded with real-world data evidence, which data constraints may suggest. However, real-world graphs may also contain missing edges, making potential data constraints trivially satisfied or not applicable. For example, the data constraint  $\varphi_2$  cannot be directly enforced to ground “review” relation due to the missing edge “pre-condition” from user to “Insomnia”. This calls for an effective method to generate subgraphs that can both clarify the output of inference queries, and also make it grounded in the evidence suggested by data constraints.

**Example 2:** Following example 1, a more desired provenance structure for the molecular property analysis scenario, as shown in Fig. 1, can be a subgraph  $G_{g_1}$  enclosed by the pink dotted oval, including a grounded edge (blue dotted) added to close the 6-atom ring, reconstructing the benzene ring. This makes the output better aligned with the chemically valid evidence. Similarly, a more comprehensive interpretation of the purchase intention of the user can be captured by a graph that includes the red edges from  $G_{s_2}$ , and a new edge (blue dotted), revealing the relation between the user and insomnia. This helps the analyst to find a more reasonable recommendation as “Melatonin” (sleep aid), better aligned with the true purchase intention, enabled by the grounded interpretation. On the other hand, as  $\varphi_2$  cannot be applied to enforce the missing edge, how to derive such a structure?  $\square$

This calls for cost-effective methods to generate provenance data for graph inference queries in possibly incomplete graphs. Likening data provenance that compute provenance structures as influential fraction of the input data for structured queries [5], we are interested in developing a Why-provenance mechanism for Inference queries over GNNs. Such provenance structure *should* capture not only the model’s internal decision-making logic but also be contextualized by external constraints suggested by data constraints. In this paper, we propose the following problem.

**Constraint-based Provenance for graph inference.** Given a graph  $G$  with a test node  $v_t$  of interest, a GNN  $M$ , an inference query  $Q(M, v_t, G)$  that returns the model’s output over  $v_t$  on  $G$ , and a set of data constraints  $\Sigma$ , the goal is to derive a cohesive (connected) graph  $G_g$  that can suggest

- a “witness”  $G_s$  for the model output at  $v_t$ , i.e., the result of  $Q(M, v_t, G)$ , that specifies the edges necessarily needed to reconstruct the latter, and
- satisfies some data constraint  $\varphi \in \Sigma$ , hence “ground”  $G_s$  to validated network patterns with real-world meanings (e.g., chemical compounds, physical system, social communities, or metabolic structures).

Note that  $G_g$  may *not* necessarily be a subgraph of  $G$ .

**Related work.** We summarize related work below.

*Provenance for graph queries.* The concept of provenance in data management refers to the lineage or derivation history of query results, aiming to trace how and why a particular output was produced from a given input [8]. These approaches have primarily focused on provenance for structured queries.

Recent work has extended provenance to graph queries. [31] formalize and answer why-questions for subgraph entity search over attributed graphs. They propose a general query rewriting framework to address why, why-not, why-rank questions, and develop practical algorithms with approximation guarantees for generating informative rewrites. [40] investigates why-questions in the context of structural graph clustering, focusing on user inquiries about unexpected clustering results. Their work aims to refine clustering configurations to better align with user expectations, thus improving the interpretability of clustering outputs in graph applications. [39] addresses why-not questions in radius-bounded k-core searches over geo-social networks, helping users understand why certain expected nodes are absent from the results by refining query parameters under spatial and structural constraints.

While these works provide valuable insights into provenance for graph queries, they are fundamentally designed for symbolic pattern search and graph mining tasks. They do not address the question of why a graph machine learning model—such as a GNN—produces a particular prediction, nor do they explain the decision-making process of such models. In contrast, our work focuses on post-hoc provenance for graph inference, aiming to explain model predictions by exploiting graph data constraints.

*GNN explanation.* A growing body of research has focused on explaining the predictions of GNNs, aiming to identify the most influential substructures that contribute to a model’s output. Existing post-hoc methods can be broadly categorized into perturbation-based (e.g., GNNExplainer [24, 35]), surrogate-based (e.g., PGExplainer [33]), and causality-inspired approaches (e.g., GEM [21], RCEExplainer [34], OrphicX [22]). Recent work has attempted to improve interpretability through more structured reasoning, yet mostly stop at representing subgraphs into logic rules rather than exploiting the latter [3, 20, 36]. While these methods differ in how they estimate subgraph importance—via masking, generative modeling, or causal inference—they all primarily focus on optimizing for behavioral fidelity, i.e., how well the explanation aligns with

the model’s output. However, they do not guarantee that the explanations conform to any user-defined or domain-specific structural constraints, and may thus yield semantically invalid or misleading interpretations in real-world applications.

Unlike these methods, our approach ensures the quality of provenance information not only in terms of factual and counterfactual validity—*i.e.*, whether the extracted evidence faithfully supports or refutes a given prediction—but also in terms of alignment with meaningful graph patterns. Specifically, the identified subgraphs are required to satisfy a set of declarative structural constraints, ensuring that the explanations are both faithful to model behavior and well-grounded in domain-relevant graph semantics. To the best of our knowledge, this is the first approach to explicitly integrate declarative graph constraints into post-hoc GNN inference analysis.

## 2 INFERENCE, CONSTRAINTS AND WITNESS

### 2.1 Graph Inference Queries

**Graphs.** A graph  $G = (V, E)$  has a set of nodes  $V$  and a set of edges  $E \subseteq V \times V$ . Each node  $v$  has a type  $v.l$ , and carries a tuple  $v.T$  of attributes and their values. The size of  $G$ , denoted as  $|G|$ , is the total number of its edges ( $|G| = |E|$ ).

Given a node  $v$  in  $G$ , the  $L$ -hop neighbors of  $v$ , denoted as  $N^L(v)$ , refers to the set of all the nodes in the  $L$ -hop of  $v$  in  $G$ . The  $L$ -hop subgraph of a set of nodes  $V_s \subseteq V$ , denoted as  $G^L(V_s)$ , is the subgraph induced by the node set  $\bigcup_{v_s \in V_s} N^L(v_s)$ .

**Graph Neural Networks.** GNNs [11, 18] comprise a well-established family of deep learning models tailored for analyzing graph-structured data. GNNs generally employ a multi-layer message-passing scheme as shown in Equation 1.

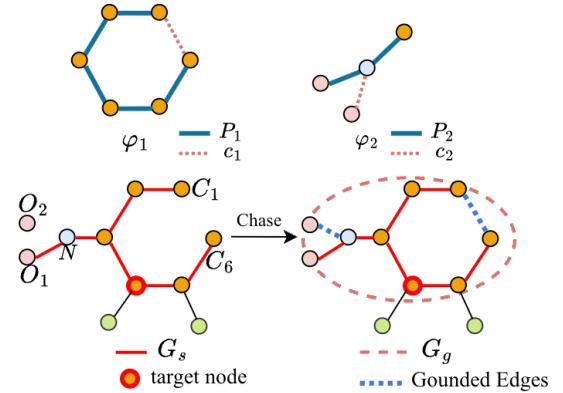
$$\mathbf{H}^{(l+1)} = \sigma(\tilde{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}) \quad (1)$$

where  $\mathbf{H}^{(l+1)}$  is the matrix of node representations at layer  $l$ , with  $\mathbf{H}^{(0)} = \mathbf{X}$  being the input feature matrix.  $\tilde{\mathbf{A}}$  is a normalized adjacency matrix of an input graph  $G$ , which captures the topological feature of  $G$ .  $\mathbf{W}^{(l)}$  is a learnable weight matrix at layer  $l$  (a.k.a. “model weights”).  $\sigma$  is an activation function.

The *inference process* of a GNN  $M$  with  $L$  layers takes as input a graph  $G = (X, \tilde{\mathbf{A}})$ , and computes the embedding  $\mathbf{H}_v^{(L)}$  for each node  $v \in V$ , by recursively applying the update function in Equation 1. The final layer’s output  $\mathbf{H}^{(L)}$  (a.k.a. “output embeddings”) is used to generate a task-specific *output*, by applying a post-processing layer (*e.g.*, a softmax function). We denote the task-specific output as  $M(v, G)$ , for the output of a GNN  $M$  at a node  $v \in V$ .

**Deterministic Inference.** A GNN  $M$  has a *fixed* inference process if its inference process is specified by fixed model parameters, number of layers, and message passing scheme. It has a *deterministic* inference process if  $M(\cdot)$  generates the same result for the same input. We consider GNNs with fixed, deterministic inference processes for consistent and robust performance in practice.

**Inference Query.** An *inference query*  $Q$  is a triple  $Q = (M, v_t, G)$  where  $M$  is a fixed, deterministic GNN model,  $v_t$  is a target node in a test graph  $G$ . The result of an inference query  $Q$  is the output  $\mathbf{H}_v^{(L)}$  of the model  $M$  via inference computation. As  $M$  is fixed and



**Figure 2: Two data constraints are illustrated:  $\varphi_1 = (P_1, c_1)$  enforces a carbon ring, and  $\varphi_2 = (P_2, c_2)$  completes a missing bond between nitrogen and oxygen. During Chase, a subgraph  $G_s$  matching  $P_1$  and  $P_2$  (red edges) is identified, and the missing bonds (blue dotted) are added to satisfy  $c_1$  and  $c_2$ , producing  $G_g \models \{\varphi_1, \varphi_2\}$  that satisfies both constraints and restores consistent molecular structure.**

indicate, the output of  $Q$  is uniquely determined by  $G$  and a graph model  $M$  as a function  $G \rightarrow \mathbf{H}_v^{(L)}$ .

A query workload  $Q = (M, V_T, G)$  defined on a set of test nodes  $V_T \subseteq V$  is a set of inference queries  $Q(M, v_t, G)$  ( $v_t \in V_T$ ).

**Example 3:** Consider the mutagenic molecule  $G_1$  in Fig. 1. Let  $M$  be a 3-layer GCN trained for node classification, where each node represents an atom and the task is to predict whether an atom is *mutagenic*. An inference query  $Q(M, v_t, G_1)$  targets the highlighted atom node  $v_t$ , asking the model to compute its final-layer representation  $\mathbf{H}_{v_t}^{(3)}$  and corresponding label. The model aggregates information from the 3-hop neighborhood of  $v_t$  to compute its final representation, which is then used to predict the atom’s mutagenicity. A workload  $Q(M, V_T, G_1)$  thus consists of all such per-atom inference queries for the test nodes  $V_T$  in the molecule. □

### 2.2 Graph Data Constraints

We consider a class of graph data constraints (or simply “constraints”). A constraint is in the form of a pair  $(P, c)$ .

- (1)  $P$  is a conjunction of triple patterns (a graph pattern), where a triple pattern is in the form of  $\langle u_s, u_p, u_o \rangle$ . Each pattern node  $u_s$  represents a class of typed entity, and can be associated with a variable  $X_s$  and a conjunction of literals in the form of  $u_s \text{ op } a$ , where  $a$  is a constant, and  $\text{op}$  is a comparison operator in  $\{=, >, <, \geq, \leq, \neq\}$ . The predicate  $u_p$  specifies the relation connecting  $u_s$  and  $u_o$ , while the object  $u_o$  denotes either a literal value or another typed entity.
- (2)  $c$  is a single triple pattern (“consequent”) in the same form as in (1), and involves at least one node (variable) seen in  $P$ .

**Semantics.** Given a data constraint  $\varphi$ , we say a graph  $G$  *satisfies*  $\varphi$  at a node  $v$ , denoted as  $G \models \varphi$ , if there exists a subgraph  $G_g$  of  $G$  that contains  $v$ , and (1) matches the graph pattern  $P$  in terms of subgraph isomorphism, and (2) has at least an edge that also satisfies the consequent  $c$ .

In practice, a consequent may enforce *Type*: that entities must have a certain type (e.g., verifying that “*a person is identified as an actor or director*” in a movie-related query); *Value Binding*: Restricting values for literals (e.g., ensuring that a date falls within a specific range or that a number must be above a threshold); or *Structural Constraints*: Declaring relationships with cardinality constraints (e.g., *a director must have directed at least one film*”).

Given a set of constraints  $\Sigma$ , the graph  $G$  satisfies  $\Sigma$ , denoted as  $G \models \Sigma$ , if for every constraint  $\varphi \in \Sigma$ ,  $G \models \varphi$ .

We say a graph data constraint  $\varphi$  is *trivial*, if  $c \in P$ . Given a pair of graph data constraints  $(\varphi, \varphi')$ , we say  $\varphi = (P, c)$  implies  $\varphi' = (P', c)$ , if  $P \subseteq P'$ . In our work, we assume a set of graph data constraints  $\Sigma$  that are non-trivial.

**Remarks.** The above data constraints can be considered as a class of tuple (edge) generating dependencies (tgds) over a structured representation  $G$ . We also showcase (see Appendix) that this formulation can be specified to express graph association rules [13], graph functional [14], or generation dependencies [30], or SHACL [27].

**Enforcement.** Chase [4, 6] has been a powerful tool for query expressiveness, constraint reasoning and enforcement. Given a set of data constraints  $\Sigma$  and a dataset  $D$ , a Chase process is a general technique that can transform  $D$  such that  $D \models \Sigma$ . When  $\Sigma$  is used to express a query  $Q$  (hence a set of query constraints), Chase can be used for evaluating conjunctive queries. If  $\Sigma$  is defined as data quality rules or constraints [25], Chase enforces  $\Sigma$  as a data repairing process. Applying Chase twice yields Chase&BackChase, a technique used for query optimization or query suggestion [10].

**Example 4:** Consider  $G_s$  Fig. 2, and let data constraint  $\varphi_1 = (P_1, c_1)$  and  $\varphi_2 = (P_2, c_2)$ , where  $P_1 = \{\langle C_i, bond, C_{i+1} \rangle \mid 1 \leq i \leq 5\}$ ,  $c_1 = \langle C_6, bond, C_1 \rangle$ ,  $P_2 = \{\langle N, bond, O_1 \rangle, \langle N, bond, C \rangle\}$ , and  $c_2 = \langle N, bond, O_2 \rangle$ . Via a Chase process, a subgraph  $G_s$  matching  $P_1$  and  $P_2$  (red edges) is identified, and the missing bonds  $c_1$  and  $c_2$  (blue edges) are enforced, generating a graph  $G_g \models \{\varphi_1, \varphi_2\}$  that satisfies both constraints. The enriched graph suggests a chemically consistent molecular structure, as required by the constraints.  $\square$

### 2.3 Witnesses for Graph Inference Queries

Provenance refers to the computation of data that clarifies the output of a query result. To characterize provenance for graph inference queries, we introduce a notation of witness graphs.

**Witness graph.** Given an inference query  $Q = (M, v_t, G)$  over a test node  $v_t$  and a graph ML model  $M: G \rightarrow H_v^{(L)}$ , a subgraph  $G_s$  of  $G$  is a *witness* of the result of  $Q$ , if  $G_s$  contains  $v_t$ , and satisfies one of the two conditions:

- $Q(M, v_t, G) = Q(M, v_t, G_s)$  (“factual”); or
- $Q(M, v_t, G) \neq Q(M, v_t, G \setminus G_s)$  (“counterfactual”)

**Example 5:** Given the inference query  $Q(M, v_t, G_1)$  in Example 3, the subgraph highlighted in Fig. 1 (red edges) serves as a *witness* for the prediction on  $v_t$ . The witness corresponds to the carbon–nitrogen fragment that directly influences the *mutagenic* classification of  $v_t$ : the 3-layer GCN  $M$  aggregates information from the 3-hop neighborhood of  $v_t$  to compute  $H_{v_t}^{(3)}$ . Removing any red edges would change the predicted label *i.e.*, *is\_mutagenic:“0”*.  $\square$

A witness for graph inference, likening its counterpart in Why-provenance [5], refers to a graph that is necessary to reconstruct the inference result of  $Q(M, v_t, G)$  (as a factual witness), or change the latter if removed (as a counterfactual witness). A witness can be explicitly generated by a post-hoc “explainer” that ensure a factual [35] or a counterfactual witness [23], or be implicitly induced from a learned masked adjacency matrix [24].

## 3 GROUNDING WITNESSES WITH CONSTRAINTS

As remarked earlier, a witness alone may not be aligned to meaningful real-world structures as it may only be faithful to the model [35]. We next describe how data constraints can be exploited to “ground” witnesses. We start by introducing a notion of  $k$ -grounded witness.

**$k$ -Grounded Witness.** Given a data constraint  $\varphi$ , a witness  $G_s$  of the result of  $Q(M, v_t, G)$ , and a number  $k$ , we say  $G_s$  is  $k$ -grounded by  $\varphi$ , denoted by  $G_s \models_k \varphi$ , if there exists a graph  $G_g = G_s \oplus \Delta E$ , *i.e.*, obtained by adding at most  $k$  new edges  $\Delta E$  that are not in  $G$ , such that (1)  $G_g$  contains  $\Delta E$  and  $G_s$ , and (2)  $G_g \models \varphi$  ( $|\Delta E| \leq k$ ). We refer to the edges in  $\Delta E$  as *grounded edges*.

We remark that the symbol  $\oplus$  means “extended” but not “union”, *i.e.*,  $G_g$  may contain  $G_s$ , a set of new edges  $\Delta E$  not in  $G$ , and a set of edges in  $G$  but not in  $G_s$  – to make  $G_g$  a cohesive, connected graph.

A witness  $G_s$  is  $k$ -grounded by a set of data constraints  $\Sigma$ , if for every constraint  $\varphi$ ,  $G_s \models_k \varphi$ . Ideally,  $G_s$  is 0-grounded, *i.e.*,  $G_s \models \varphi$ .

A  $k$ -grounded witness  $G_s$  is *minimal*, if (1)  $G_s$  is a minimal witness, and (2)  $\Delta E$  is a minimal set of edges that ensures  $G_s \models_k \varphi$ .

**Example 6:** Consider the constraint set  $\Sigma = \{\varphi_1, \varphi_2\}$  in Fig. 2, where  $\varphi_1$  enforces the carbon ring closure and  $\varphi_2$  regulates the nitrogen–oxygen bonding pattern. Let  $G_s$  be the witness (red bold edges) that matches  $P_1$  and  $P_2$ , and let  $G_g = G_s \oplus \Delta E$  be the extended graph enclosed by the pink dotted boundary. Here  $\Delta E$  denotes the blue dashed links not in the original  $G$ , which correspond to the consequents  $c_1$  and  $c_2$ , respectively. Specifically, the added edge  $(C_6, C_1)$  enables  $G_g \models \varphi_1$  by closing the carbon ring, and  $(N, O_2)$  ensures  $G_g \models \varphi_2$  by completing the nitrogen–oxygen linkage. Hence  $G_s \models_1 \varphi_1$  and  $G_s \models_1 \varphi_2$ , indicating that  $G_s$  is 1-grounded by  $\Sigma$ .  $\square$

**Applications.**  $k$ -grounded witnesses  $G_s$  are desirable as justified below. (1) High interpretability. It readily suggests critical fraction  $G_s$  of the original graph  $G$  that are responsible to  $Q(M, v_t, G)$ , and an extended counterpart  $G_g$  to ensure a co-occurrence of matches of  $P$  and  $c$ , hence directly grounding  $G_s$  to genuine, real-world data evidence as specified by  $\varphi$ . The constraints  $\varphi$ , in turn, contributes to rule-based GNN interpretation as logical rules. (2) Data preparing and refinement for graph learning. While conventional graph enrichment focuses on maximizing data completeness rather than “ML-aware”, the additional edges  $\Delta E$  readily suggest “how”  $G$  should be modified to include missing data, that are specifically relevant to decision making process of  $M$ . This also indicates useful training and testing triples for GNNs. (3) Adversarial graph learning. The  $\Delta E$  and  $G_s$  together indicate “vulnerable” area that may be attacked or poisoned to affect GNN  $M$ ’s output, hence robust training can be localized to avoid expensive defense cost.

**Quality Measures of Grounded Witnesses.** As there are many witnesses that can be  $k$ -grounded by a constraint  $\varphi$ , we introduce a quality function to find high-quality witnesses.

**Conciseness & Minimality.** A witness should be concise, following Occam’s Razor, by retaining as few edges as possible. (1) A witness  $G_s$  is *minimal* for an inference query  $Q(M, v_t, G)$ , if any of its subgraph obtained by removing an edge is not a witness of  $Q(M, v_t, G)$ . (2) Consider  $G_{v_t}^L$ , the  $L$ -hop subgraph of a test node  $v_t$ . The conciseness of  $G_s$  is defined as  $\text{conc}(G_s) = 1 - \frac{|G_s|}{|G^L(v_t)|}$ . We aim to find minimal witnesses with higher conc scores.

**Alignment cost.** Ideally, a witness  $G_s$  should be 0-grounded by a data constraint  $\varphi$  - that is,  $G_s \models \varphi$ . Given  $\varphi$ , two  $k$ -grounded, minimal witness  $G_{s_1}$  and  $G_{s_2}$  that of the same size, we further break tie with the size of the minimal set  $\Delta E$  required to have  $G_s$   $k$ -bounded by  $\varphi$ , denoted as an alignment penalty. We introduce a function  $\text{aln}(G_s, \varphi) = 1 - \frac{|\Delta E|}{k}$ . Higher score indicates smaller alignment penalty. Here  $|\Delta E| \in [0, k]$ . A 0-grounded  $G_s$  has  $\text{aln}(G_s, \varphi) = 1$ .

**Coverage.** Given a set of constraints  $\Sigma$ , a witness  $G_s$  ideally is  $k$ -grounded by all  $\varphi \in \Sigma$ . We define a coverage measure of  $G_s$ , denoted as  $\text{cov}$ , as  $\text{cov}(G_s) = \frac{|\Sigma(G_s, k)|}{|\Sigma|}$ , where  $\Sigma(G_s, k) = \{\varphi | G_s \models_k \varphi, \varphi \in \Sigma\}$ , i.e., the subset of  $\Sigma$  by which  $G_s$  is  $k$ -grounded.

**Overall Quality.** We favor minimal witnesses that are concise, consistent, with small alignment cost, measured by a function  $F(G_s)$ :

$$F(G_s) = \alpha \cdot \text{conc}(G_s) + \beta \cdot \text{aln}(G_s) + \gamma \cdot \text{cov}(G_s)$$

where  $\alpha, \beta, \gamma \geq 0$  are tunable weights.

**An Axiomization analysis.** We justify  $F(\cdot)$  with an axiomization analysis below. For any inference query  $Q(M, v_t, G)$ , let  $G_s^*$  be the optimal witness that maximizes  $F(G_s^*)$ .  $F(\cdot)$  has the following properties. (1) **Scale invariance.**  $G_s^*$  remains to maximizes  $F$  regardless how  $\text{conc}$ ,  $\text{aln}$  and  $\text{cov}$  are scaled by some constant. This ensures the invariance of  $G_s^*$  regardless of how the user preference  $(\alpha, \beta, \gamma)$  changes. (2) **Non-monotonicity.**  $F(G_s)$  is not necessarily less than  $F(G'_s)$  if  $|G_s| \leq |G'_s|$ . Indeed, larger witnesses does not necessarily indicate better  $F$  scores. (3) **Independence.**  $F$  is only determined by the nodes and edges in  $G_s$ . No information from entities or edges not seen in  $G_s$  can affect its quality. This property aligns well with the need to find witnesses in a pragmatic “semi-closed world” assumption, striking a balance between a challenging, if solvable, open-world setting ( $G$  is infinite) and a rigorous, yet an overkill, close world assumption, where  $G$  has no missing edges.

**Problem Statement.** Given an inference query  $Q(M, v_t, G)$ , a budget  $k$ , and data constraints  $\Sigma$ , the problem of *Top-K witness generation* computes a  $K$ -set of witnesses  $\mathcal{G}_s$  of  $Q$ , such that (1) each  $G_s \in \mathcal{G}_s$  is a minimal  $k$ -grounded witness by some constraint  $\varphi \in \Sigma$ , and (2)  $\mathcal{G}_s$  optimizes the following aggregated quality score:

$$\mathcal{G}_s = \arg \max_{|\mathcal{G}'_s|=K} \sum_{G_s \in \mathcal{G}'_s} F(G_s)$$

This problem is non-trivial as verified by a hardness result below (see detailed hardness analysis in [1]).

**Theorem 1:** *The witness generation problem is  $\Pi_2^P$ -hard for an inference query  $Q(M, v_t, G)$  with fixed  $G$  and  $M$ .*  $\square$

**Proof sketch:** We construct a polynomial-time reduction from the min-max Clique problem [28], where the single constraint has a clique as  $P$  and  $c$  is the empty set. The problem instance can be reduced to an instance of witness generation that aims to find if there exists a 0-grounded witness from a finite set of graphs (which are all witnesses for a trivial GNN that assigns a fixed label to all the nodes in  $G$ ) that must contain a largest clique with size at least  $k$ . This requirement can be encoded by a finite set  $\Sigma$  of at most  $k$  constraints, with patterns  $P$  as a size  $i$  clique, and  $i \in [1, k]$  for the  $i$ -th constraint. As  $G$  and  $M$  are fixed (with  $L$  number of layers), there are polynomial number of subgraphs (a “candidate”)  $G_c$  in  $G$  to be verified for witnesses. For each subgraph  $G_c$ , the reduction invokes a procedure *VerifyWitness* that computes  $Q(M, v_t, G_c)$  and  $Q(M, v_t, G \setminus G_c)$  to confirm if  $G_c$  is a witness, both in polynomial time, as graph inference is in PTIME [7, 37]. If there is a witness that satisfies  $\Sigma$ , it indicates a graph exists from a set of input graphs that contains a largest clique with size  $k$ ; and vice versa. As min-max Clique is known to be  $\Pi_2^P$ -complete, the hardness follows.  $\square$

We next present practical algorithms to generate  $k$ -grounded witnesses for provenance of graph inference under constraints.

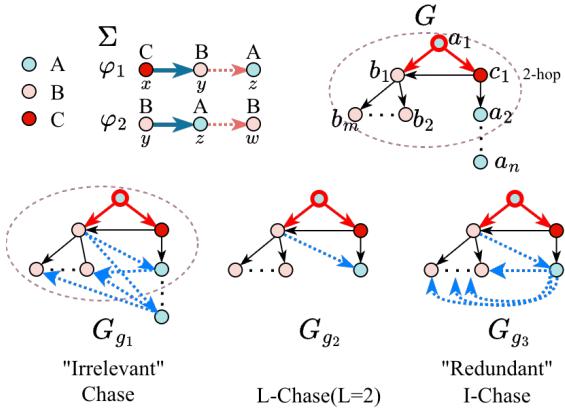
**A naive solution.** We start with a simple algorithm, denoted as *naiveChase*, that extends chase with graph pattern matching and graph inference to generate  $k$ -grounded witnesses. The procedure follows a stacked “enforce-and-provenance” process.

(1) Given a graph  $G$ ,  $\Sigma$ , and a local budget  $k$ , *naiveChase* first enforces  $\Sigma$  over  $G$  via a fixed-point computation. This is done by a variant of Chase algorithm, denoted as *I-Chase*, which iteratively enrich an edge  $(v, v')$  as a match of  $c(u, u')$  whenever a subgraph match of  $P$  is identified in  $G^L(v_t)$ , and terminates when no  $\varphi$  can be enforced. The result below ensures the generation of a finite graph  $G'$ , such that  $G' \models \Sigma$ .

**Lemma 2:** *Given a graph  $G$  and a set of constraints  $\Sigma$ , (1) I-Chase always terminates after a finite steps of enforcement, and (2) The computation has the Church-Rosser Property.*  $\square$

(2) For each  $\varphi = (P, c)$ , and each subgraph  $G_s$  that matches  $P$ , it then invokes a verification process *VerifyWitness* (Section 3) to check if  $G_s$  is a witness of  $Q(M, v_t, G)$ , and generate  $K$  minimal subgraphs  $G_g$  of  $G'$  that are witnesses. In this scenario, each  $G_s$  is ensured to be a 0-grounded witnesses by some constraint, given the observation that if  $G \models \varphi$ , then for any subgraph  $G_s$  of  $G$  that contains a match of  $P$ ,  $G_s \models \varphi$ .

While *naiveChase* can derive a set of grounded witnesses (see Analysis in [1]), it remains infeasible in practice for large  $G$  with missing edges. (1) The enforcement of  $\Sigma$  over the entire  $G$  is inheritability expensive due to an exhaustive enumeration of subgraphs that matches  $P$  from the data constraints in terms of subgraph isomorphism. The latter is already computationally hard and expensive in practice. (2) Another issue is that the generated witnesses  $G_g$  may not be “faithfully” reflecting the contribution of the *original* graph  $G$  in decision making of the model  $M$ , as the inference is conducted post-hocly on a refined graph with enforced edges that do not exist in  $G$ . (3) The witnesses may not be minimal, and contain redundant edges that are not a part of matches of  $P \cup c$ .



**Figure 3: Generating grounded witnesses (Inf-Chase); unlike I-Chase as a fixpoint computation, L-Chase reasons with  $\Sigma$  up to  $L$  times from a set of “source” constraints  $\Sigma_s$ .**

We next introduce efficient algorithms, to generate  $k$ -grounded witnesses without an exhaustive enforcement of  $\Sigma$  over  $G$ .

#### 4 WITNESS GENERATION: OVERVIEW

We introduce a general algorithm, denoted as Inf-Chase.

**Outline.** The foundation of our computational idea is a variant of Chase&Backchase algorithm. It undergoes two phases below.

- (1) a **Chase** phase: which enforces  $\Sigma$  on the  $L$ -hop neighbor graph  $N^L(v_t)$  of  $v_t$ , via a procedure **L-Chase**, a variant of I-Chase that performs a bounded rounds of reasoning (see “L-Chase”); and
- (2) a **Backchase** phase, applied to enforce an *inverse* set of  $\Sigma$ , to make each witness  $G_s$   $k$ -grounded by  $\Sigma$ . This is supported by a procedure **L-Backchase**, which enforces an input inverse constraint set  $\Sigma_b \subseteq \Sigma^{-1}$  on  $G_s$ , with an edge update budget  $k$ , whenever possible, to obtain a graph  $G_g$  that contains a  $k$ -grounded witness  $G_s$ . Here each data constraint  $\varphi^{-1} = (c, P)$  in  $\Sigma_b$  is an inverse counterpart of some  $\varphi = (P, c)$  in  $\Sigma$ .

Putting together, the process ensures to derive a  $K$ -set of graphs  $\mathcal{G}_g$ , such that each graph  $G_g \in \mathcal{G}_g$ , (1) contains a  $k$ -grounded witness  $G_s$ , and (2)  $G_g \models \varphi_b^{-1} \subset \Sigma$ . The latter holds due to the co-occurrence of matches of both  $c$  and  $P$  in  $G_g$ . The overall computation, in a nutshell, repeat two steps below up to  $L$  times.

$$\mathcal{G}_s := \text{I-Chase}(\Sigma, G^L(v_t), 0); \quad \mathcal{G}_g := \text{I-Chase}(\Sigma^{-1}, \mathcal{G}_s, k).$$

**Procedure L-Chase.** Unlike I-Chase, L-Chase differs in the following. (a) Rather than a fixpoint computation as in I-Chase, it performs up to  $L$ -rounds of Chase, starting at  $v_t$  and a set of “source” constraints  $\Sigma_s$  with pattern nodes having  $v_t$  as a match, to avoid “over grounding” the output  $Q(M, v_t, G)$  with real-world entities that are too far or irrelevant to  $v_t$ . (b) It extends I-Chase to directly derive a set  $\mathcal{G}_s$  of minimal, 0-grounded witnesses (by setting  $k=0$ ). Specifically, for each  $\varphi = (P, c) \in \Sigma$ , L-Chase  $(\Sigma, G, k)$  follows a graph pattern mining process to generate a set of candidate subgraphs  $\mathcal{G}_c$  via an edge growing strategy, and invokes procedure VerifyWitness to retain those that are witnesses for  $Q(M, v_t, G)$ . Note that no new edges are enforced in this stage.

**Procedure L-Backchase.** L-Backchase If  $k > 0$  (“Backchase”), then procedure L-Backchase find all  $\varphi^{-1} = (c, P)$  with consequent  $c$  having an edge match  $e_c$  in  $G_c$ , and insert a minimal set  $\Delta E$  of at most  $k$  new edges to  $e_c$ , to “enforce” at least an occurrence of a match of  $P$ . As such,  $G_c$  is  $k$ -grounded by  $\varphi^{-1}$ . Including a connected subgraph  $G_g$  with  $G_c$  and  $\Delta E$  ensures to generate a set of minimal witness  $\mathcal{G}_g$ . This step introduces at most  $|\Sigma|k$  new edges.

**Example 7:** Consider the constraint set  $\Sigma = \{\varphi_1, \varphi_2\}$  shown in Fig. 3. All the node  $a_i$  in  $G$  has a same type A in capital ( $a_i.l = 'A'$ ); similar for other nodes.  $\varphi_1 : (P_1, c_1)$  (with node variables  $x, y, z$ ) enforces that any node  $y$  of type B with an edge from a node  $x$  of type C (*i.e.*,  $C \rightarrow B$ ) should also have a link to a different node  $z$  of type A (*i.e.*,  $B \rightarrow A$ ), and  $\varphi_2 : (P_2, c_2)$  enforces that whenever  $B \rightarrow A$  holds, a link  $A \rightarrow B'$  should also exist. On the right-hand side of Fig. 3 we shows a graph  $G$ , where the red bold edges denote a witness  $G_s$  covering the one-hop neighborhood of the target node  $v_t$ , and the dashed oval marks its 2-hop receptive field (given  $L = 2$ ). We observe the following.

- (1) As  $P_1$  is matched, a Chase, as a fixpoint process, enforces  $c_1$  to insert edges between  $b_2$  and any  $a$  node regardless of how large  $n$  is ( $b_2 \rightarrow a_2, \dots, a_n$ ). Most of edges fall outside the receptive field of  $v_t$  and thus become irrelevant (see  $G_{g_1}$ ).
- (2) In contrast, I-Chase restricts reasoning within  $G^L(v_t)$ , hence prevent  $a_i$  that is too far from  $a_1$  to be considered for grounding. However, in the second round of chase,  $\varphi_2$  is triggered, enforcing redundant, multiple semantically-equivalent structures (*e.g.*,  $a_2 \rightarrow b_2, \dots, b_m$ ), leading to an excessive number of “over-grounded” graphs that are all isomorphic.
- (3) L-Chase strikes a balanced alternative: by performing up to just one more round of reasoning (as  $L=2$ , and  $G_s$  already include all hop-1 neighbors) within  $G^L(v_t)$ , it suffices to chase one round to produce an informative and grounded result ( $G_{g_2}$ ) with less irrelevant or redundant edges. This illustrates the need to have a bounded hop, and bounded rounds of chase to avoid irrelevant and redundant provenance structure for grounding the witnesses.  $\square$

**Correctness.** We show that Inf-Chase correctly computes a set of graphs  $\mathcal{G}_g$ , such that each  $G_g$  contains a minimal  $k$ -grounded witness  $G_s$  by some  $\varphi \in \Sigma$ . We observe two invariants during the process of Inf-Chase. (1) The Chase phase ensures to identify a set of subgraphs  $\mathcal{G}_s$  of  $G^L(v_t)$ , such that for each subgraph  $G_s \in \mathcal{G}_s$ , (a)  $G_s \models \varphi$  for some  $\varphi = (P, c) \in \Sigma$  (without introducing new edges), and (b)  $G_s$  has been verified to be either a factual or a counterfactual witness for  $Q(M, v_t, G)$ , guaranteed by the correctness of I-Chase. (2) As  $G_s \models \varphi$ , it contains an edge  $e_c$  as a match of  $c$  in  $\varphi$ . For any  $\varphi'$  with  $P' \in c$ , its inverse form  $\varphi'^{-1} = (c, P')$ . The backchase phase I-Chase  $(\varphi'^{-1}, G_s, k)$  makes  $G_s$   $k$ -grounded, by identify all applicable  $\varphi'^{-1}$  over  $G_s$  and enforcing  $(c, P')$  over all witnesses  $G_s \in \mathcal{G}_s$ , treating  $\varphi'^{-1}$  as a tuple(“edge”)-generation constraint on triples of  $G^L(v_t)$ . By definition, each  $G_g$  derived successfully ensures to contain  $G_s$  as a  $k$ -grounded witness for  $\varphi' \in \Sigma$ . That is,  $G_g = \text{I-Chase}(\varphi'^{-1}, G_s, k) = \text{I-Chase}(\varphi'^{-1}, \text{I-Chase}(\varphi, G^L(v_t), 0), k)$ , and the latter composite function is exactly computed by Inf-Chase in the two-phase Chase & Backchase process.

---

**Algorithm 1 : ApxlChase**


---

**Input:** Constraint set  $\Sigma$ , query  $Q(M, G, v_t)$ , integers  $L$  and  $K$ ;  
**Output:** the current top- $K$   $k$ -grounded witness set  $W_K$ .

```

1: construct  $G_\Sigma := \bigcup_{\varphi \in \Sigma} P$ ; initializes  $\mathcal{M}$ ; set  $\mathcal{G}_g := \emptyset$ ,  $\mathcal{G}_c := \emptyset$ ,
2:  $W_K := \emptyset$ ;  $\Sigma_s := \emptyset$ ;  $\Sigma_b := \emptyset$ ;
3: for  $\ell = 1$  to  $L$  do
4:    $\Sigma_s := \{\varphi | u.l = v_t.l, u \text{ is from } \varphi.P; \varphi \in \Sigma\}$ ;
5: /* Chase phase */
6:    $\mathcal{G}_s := \mathcal{G}_s \cup \text{L-Chase}(\Sigma, \Sigma_s, G^L(v_t), G_\Sigma, \mathcal{M}, 0)$ ;
7: /* initializes backchase */
8:   set  $\Sigma_b := \emptyset$ ;
9:   for each  $G_s \in \mathcal{G}_s$  and each  $\varphi \in \Sigma$  do
10:    if  $G_s \models \varphi$  then  $\Sigma_b := \Sigma_b \cup \varphi^{-1}$ ;
11: /* backchase phase */
12:    $\mathcal{G}_g := \mathcal{G}_g \cup \text{L-Backchase}(\Sigma, \Sigma_b, \mathcal{G}_s, G_\Sigma, \mathcal{M}, k)$ ;
13: /* maintain current Top-K grounded witness set*/
14:    $W_K := \text{UpdateWK}(W_K, \mathcal{G}_g, K, B)$ ;
15: return  $W_K$ ;

```

---

Figure 4: Algorithm ApxlChase: Chase & Backchase with bounded enforcement and edge insertion.

**Time cost.** Both Chase and backchase phases takes  $|\Sigma|$  calls of l-Chase, a fixed-point reasoning process as a sequence of  $L$  enforcement of constraints. It takes in total  $O(|G^L(v_t)|^p)$  time to find matches of patterns from  $\Sigma$ , where  $p$  is the size of the largest pattern  $P$  a data constraint has in  $\Sigma$ . The inference cost is in  $O(L|G^L(v_t)||N^L(v_t)|)$  time [7, 37], assuming  $N^L(v_t)$  is much larger than the number of node features. The total cost of Inf-Chase is thus in  $O(|G^L(v_t)|^p) + L|\Sigma||G^L(v_t)||N^L(v_t)|$ .

While Inf-Chase ensures to generate all  $k$ -grounded witnesses  $\mathcal{G}_g$ , (1)  $\mathcal{G}_g$  may not be optimizing the quality as desired; and (2) it remains to be expensive due to an exhaustive enumeration of subgraphs in  $G^L(v_t)$ . We next introduce two practical algorithms to make Inf-Chase feasible for large graphs.

## 5 ONLINE WITNESSES GENERATION

Our first algorithm, denoted as ApxlChase and illustrated in Fig. 4, specifies Inf-Chase as an online generation process with (1) a further optimization of L-Chase with cost-effective and memoization structures, to reduce excessive inference and verification cost; and (2) ensures an “any-time” quality guarantee with a small delay time.

**Theorem 3:** *There is an online algorithm that returns a top- $K$   $k$ -grounded witness, upon request time, with a delay time in  $O(|\Sigma|^2|G^L(v_t)| + L|\Sigma|^2 + |\Sigma|K^2)$ , and ensures a  $1 - \frac{1}{e}$  approximation ratio with the current optimal solution so far.*  $\square$

**Memoization Structures.** ApxlChase stores and maintains a set  $\mathcal{G}_g$  of top- $K$   $k$ -grounded witnesses in a window  $W_K$ . Besides, it also keeps track of two memoization structures below. (1)  $G_\Sigma$  is a graph pattern obtained as the union of the patterns  $P \cup c$ , encoding all the triple patterns from data constraints of  $\Sigma$ . Each edge  $e = (u_s, u_p, u_o)$  in  $G_\Sigma$  carries a set of labels  $\mathcal{L}(e) = \{i | e \text{ is in } P_i \text{ or } c_i\}$  for any  $\varphi_i = (P_i, c_i)$ , i.e.,  $e$  indicates a common triple pattern in multiple

---

**Algorithm 2 : Procedure L-Chase ( $\Sigma, \Sigma_s, G^L(v_t), G_\Sigma, \mathcal{M}, 0$ )**


---

```

1: set  $\mathcal{G}_c := \emptyset$ ;
2: for  $\varphi \in \Sigma_s$  do
3:   graph  $G_c := \text{getNext}(\varphi, G_\Sigma, \mathcal{M})$ ;
4:   if  $G_c \neq \emptyset$  then
5:      $\mathcal{G}_c := \mathcal{G}_c \cup \{G_c\}$ ; update  $G_\Sigma, \mathcal{M}$ ;
6:     if VerifyWitness( $G_c, Q(M, v_t, G)$ ) = true then
7:        $\mathcal{G}_s := \mathcal{G}_s \cup \{G_c\}$ ;
8: return  $\mathcal{G}_s$ ;

```

---

Figure 5: Procedure L-Chase

---

**Algorithm 3 : Procedure L-Backchase ( $\Sigma^{-1}, \Sigma_b, \mathcal{G}_s, G_\Sigma, \mathcal{M}, k$ )**


---

```

1: for each  $\varphi^{-1} \in \Sigma^{-1}$  do
2:   for each  $G_s \in \mathcal{G}_s$  do
3:     if  $\mathcal{M}[i].b > k$  then continue ;
4:     derive  $\Delta E$  from  $\mathcal{M}[i].P$  (“0” entries);
5:     construct and verify  $G_g$  (w. Weisfeiler Leman test);
6:     if  $G_g$  contains an occurrence of  $P_i$  then
7:        $\mathcal{G}_g := \mathcal{G}_g \cup G_g$ ;
8: return  $\mathcal{G}_g$ ;

```

---

Figure 6: Procedure L-Backchase

constraints. (2) A shared map  $\mathcal{M}$  with size  $|\varphi| \times (p+2)$ , with  $p$  the size of the largest pattern  $P$  in  $\Sigma$ , and for each entry  $\mathcal{M}[i]$ ,

- $\mathcal{M}[i].P$  is a size- $p+1$  binary vector, such that each slot  $\mathcal{M}[i].P[j] (j \in [1, p]) = 1$  (resp.  $\mathcal{M}[i].P[p+1] = 1$ ) if and only if  $\varphi_i$  has its  $j$ -th triple pattern in  $P_i$  (resp.  $\varphi_i.c_i$ ) has a non-empty edge match, in a current candidate subgraph  $G_s$  under verification; and
- (ii)  $\mathcal{M}[i].b$ , a budget number that records a lower bound of edges required to make  $G_s$   $k$ -grounded by  $\varphi_i$ .

In addition, ApxlChase keeps a set of source constraints  $\Sigma_s \subseteq \Sigma$  to start the enforcement in Chasephase; and  $\Sigma_b \subseteq \Sigma^{-1}$  to be enforced to generate  $\mathcal{G}_g$  in the backchase phase.

**Algorithm Outline.** The main driving algorithm ApxlChase, and the procedures L-Chase and L-Backchase are illustrated in Figs. 4, 5 and 6, respectively. ApxlChase first initializes the above auxiliary structures (lines 1-2). It then performs up to  $L$  rounds (line 3) of L-Chase (line 6) and L-Backchase (line 12) enforcement, consistently following Inf-Chase, and generates a batch of  $k$ -grounded witnesses  $\mathcal{G}_g$  in each round (line 12). It then invokes a procedure UpdateWK, to dynamically maintain the top- $K$  grounded witnesses via an online greedy-based replacement strategy, hence achieving an approximation ratio of  $1 - \frac{1}{e}$  for the optimal  $K$ -set of witnesses (see details of UpdateWK in [1]). The correctness and time cost follows from its counterpart of Inf-Chase.

We present the detailed analysis in [1].

**Optimization.** Algorithm ApxlChase uses several optimization strategies, with different focus.

- (1) To avoid irrelevant grounding that may not be “unfaithful” for  $Q(M, v_t, G)$ , procedure L-Chase calls a function getNex (line 3),

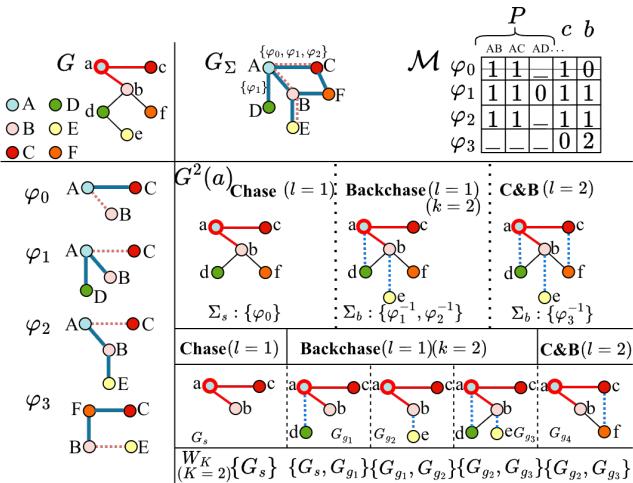


Figure 7: ApxlChase: running of C& B in 2 rounds

which follows a prioritized traversal of  $G_\Sigma$  to prefer  $\varphi$  with pattern edges that has most occurrences over  $\Sigma$ , and grows candidate subgraphs  $G_c$  with matching edges accordingly. This favors  $\varphi$  with common evidences, which are typically more credible than those rare counterparts. Other query selectivity or cardinality estimation techniques e.g., [16, 29] are readily applicable.

- (2) To reduce cost from redundant inference computation, procedure VerifyWitness performs an incremental inference strategy. Instead of always re-start inference process from scratch (line 6 of VerifyWitness), it conducts a once-for-all inference for  $Q(M, v_t, G^L(v_t))$  and caches the node embeddings. To verify if a candidate  $G_c$  is a witness, it directly loads the cached embeddings of any node  $v$  with  $G^L(v) \subseteq G^L(v_t)$ , and continue the message-passing from  $v$  to  $v_t$ , to incrementally update the node embeddings.
- (3) Procedure L-Backchase reduces enforcement cost by monitoring  $\mathcal{M}$  to optimize the enforcement cost. It maintains a lower bound of  $|\Delta E|$ , as the number of “missing edge matches”, readily recorded as the number of 0 entries in  $\mathcal{M}[i].P$ , at any time.  $\mathcal{M}$  also enables early suggestion and run-time pruning for  $\Sigma_s$  and  $\Sigma_b$  by checking if  $\mathcal{M}[i].P[d+1]$  (the consequent) has a match.
- (4) ApxlChase also copes with potential large  $\Sigma$  by removing any constraints that can be already logically implied by others, with a set of pruning rules. We present the details in [1].

These strategies are effective: they improve naive implementation naiveChase on average by one to two orders of magnitude in efficiency, as verified by our experimental study.

**Example 8:** Consider running ApxlChase on the example in Fig. 7. The original graph  $G$  (top-left) contains six typed nodes  $A-F$ , with four constraints  $\Sigma = \{\varphi_0, \varphi_1, \varphi_2, \varphi_3\}$  shown at the left. Each constraint  $\varphi_i = (P_i, c_i)$  is illustrated with its antecedent  $P_i$  (blue solid edges) and consequent  $c_i$  (pink dotted edge). The merged pattern  $G_\Sigma$  (top-right) encodes the union of  $P_i \cup c_i$  for all  $\varphi_i \in \Sigma$ . The memoization map  $\mathcal{M}$  records the grounding status of each constraint after one chase round. For example,  $\mathcal{M}[0].b = 0$  (as all the pattern edges

have a match without “0” entries) indicates that  $G_s \models_0 \varphi_0$ , hence  $\varphi_0$  is satisfied and can be removed from subsequent enforcement.

(1) During the first chase phase, the receptive field of the target node  $a$  (i.e.,  $G^2(a)$ ) has  $\Sigma_s = \{\varphi_0\}$ . In the first backchase, given budget  $k = 2$ , the inverse constraint set  $\Sigma_b = \{\varphi_1^{-1}, \varphi_2^{-1}\}$  is enforced.

(2) In the second round ( $L=2$ ), constraint  $\varphi_3^{-1}$  is also triggered, enforcing the new edge  $(c, f)$ . The bottom row shows the evolution of the top- $K$  window  $W_K$  (with  $K = 2$ ). Initially,  $G_s$  is discovered as a 0-grounded witness for  $\varphi_0$ .  $G_{g_1}$  and  $G_{g_2}$  represent  $G_s \models_1 \varphi_1$  and  $G_s \models_1 \varphi_2$ , updating  $W_K = \{G_{g_1}, G_{g_2}\}$ . Upon the arrival of  $G_{g_3}$ , ApxlChase prefers  $G_{g_3}$  given its better coverage of both  $\varphi_1$  and  $\varphi_2$ , replaces  $G_{g_1}$  with it. The current top- $K$  sets contains  $G_{g_2}, G_{g_3}$ .

(3) The process continues, and after another round of Chase & Backchase, a new graph  $G_{g_4}$  satisfies  $\varphi_3$  ( $G_s \models_1 \varphi_3$ ), yet due to a weak link to  $a$ , it does not improve the F-score ranking, hence  $W_K$  remains unchanged. The algorithm returns  $W_K$  as a final result.  $\square$

## 6 FAST WITNESSES GENERATION

Our second algorithm specifies Inf-Chase for large real-world sparse, hierarchical and tree-like networks, such as network traffic networks, power deliverable networks, epidemic networks, or citation networks [26]. In such scenarios, the genuine grounding evidences are mostly close to trees.

**Grounding witnesses as trees.** We consider a practical variant of our problem, where (1) each constraint  $\varphi = (P, c)$  in  $\Sigma$  has a tree pattern  $P \cup c$ , (2) the  $k$ -grounded witnesses  $G_s$  are weighted trees rooted at  $v_t$ , and (3)  $G_g$  is a *weight arborescences* of  $G^L(v_t)$ , i.e., a spanning tree rooted at the node  $v_t$ . This setting aligns  $G_g$  well with a full coverage of participating nodes in message passing based inference of  $Q(M, v_t, G)$ . Here a weight function  $w(e)$  for an edge  $(v, v') \in G$  can be introduced as representative functions adopted in GNN behavior and explainability analysis [24], such as embedding similarity, where  $w(e) = \text{sim}(h_u^{(L)}, h_w^{(L)}) + \varepsilon_e$ , where  $\text{sim}(a, b) = \frac{a^T b}{\|a\| \|b\|}$ , and  $\varepsilon_e$  is a small random noise (e.g., Gumbel or Gaussian) that introduces diversity across  $m$  samples of trees. Other options such as resistant distance [38], Shapley value [36], or sensitivity [19], can also be applied.

The problem remains to be NP-hard, due to the inherent intractability from subgraph isomorphism (even between a tree pattern and a directed acyclic graph) and optimal spanning tree. We next introduce a fast heuristic, denoted as HeulChase, to generate  $G_g$  that ground witnesses as maximum weighted arborescences.

**Outline.** The algorithm follows the framework Inf-Chase with a L-Chase and a L-Backchase step to generate grounded witnesses. Unlike ApxlChase, HeulChase makes the following specifications.

- (1) It uses a revised L-Chase, which generates witnesses  $G_s$  via a tree pattern matching and verification between a tree pattern  $P$  and a set of  $m$  sampled trees from graph  $G^L(v_t)$ .
- (2) It implements L-Backchase by extending each  $G_s$  to a weighted arborescence  $G_g$  rooted at  $v_t$ . To ensure  $G_s$  is contained in  $G_g$ , it shrinks  $G_s$  to a “super node”, and serves the compressed  $G^L(v_t)$  as input. This reduces the problem to computing an optimal weighted

Dataset	V	E	# node types	# attributes
BAShape [35]	2,020,000	12,055,704	4	1
Cora [26]	2708	5429	7	1433
ATLAS [2]	59,148	40,823	2	64
MUTAG [9]	17.9(avg.)	19.8(avg.)	7	7
ogbn-Papers100M [17]	111M	1.6B	40	128
Tree-Cycles [35]	67M	1.4B	5	5

**Table 1: Summary of datasets Ranked by |G|**

arborescence. L-Backchase then applies Edmonds’ algorithm [12] to derive  $G_g$  over the shrunked network.

The procedure UpdateWK remain unchanged, consistently adopting a greedy strategy to maintain top- $K$   $G_g$ .

**Analysis.** The algorithm HeulChase ensures the correct generation of provenance structures as  $k$ -grounded tree witnesses, and weighted arborescences, following the correctness analysis of Inf-Chase and the correctness of Edmonds’ algorithm. It incurs the same worst-case delay time cost as in ApxlChase. The main cost saving is due to (1) the faster tree pattern matching and inference over  $m$  sampled trees in L-Chase, which is reduced from  $|G^L(v_t)|p$  ( $p$  is the size of the largest pattern from  $\Sigma$ ) to  $O(m|G^L(v_t)| + mp^2)$ , and (2) the low cost of deriving a weighted arborescence of  $G^L(v_t)$  in each round, which is in  $O(|G^L(v_t)| \log |G^L(v_t)|)$  time [1]. The total time cost of HeulChase is thus in  $O(L \cdot m \cdot |\Sigma| + L|G^L(v_t)| \log |G^L(v_t)|)$  time.

## 7 EXPERIMENTAL STUDY

### 7.1 Experimental Settings

Using real-world and synthetic benchmark datasets, we evaluate the effectiveness, efficiency, and scalability of our algorithms.

**Datasets.** We use six real-world graphs and one synthetic graph for evaluation (summarized in Table 1). (1) *MUTAG* [9], a set of 188 chemical compound graphs. Each graph has a class label indicating whether it is mutagenic or non-mutagenic. Node feature is a 7-dimensional one-hot encoding of atom types (e.g., C, N, O). (2) *ATLAS* [2] is a cyber event graph for intrusion detection. Nodes represent system entities (e.g., process, file, connection), and edges denote data flow or dependencies. (3) *Cora* [26] is a citation network with 7 categories. Each node has a 1,433-dimensional feature vector, and edges represent citation links among papers. (4) *BAShape* [35] is a synthetic dataset generated by attaching house-shaped motifs to a Barabási-Albert base graph. A node label corresponds to structural roles (top, middle, bottom, or background). (5) *ogbn-Papers100M* [17] is a large-scale citation network with 1.6 billion citations (edges). Each paper has a 128-dimensional feature vector and is classified into one of 40 research fields. (6) *Tree-Cycles* [35] is a synthetic dataset with hierarchical graphs combining trees and cyclic structures, with 1.4 billion edges and nodes in five categories represented by 5-dimensional one-hot features.

**Constraint generation.** For each dataset, we sampled a set of test nodes  $V_T$ . For each test node  $v_t \in V_T$ , we induced  $G^L(v_t)$  ( $L=2$  by default) and sampled a set of subgraphs that contain  $v_t$ . We summarized a typed graph  $G_\Sigma$  first, and randomly sample the antecedents and consequents with matching types in  $G_\Sigma$ . We also randomly drop up to 20% of edges to simulate incomplete graphs.

**Graph Models.** We have pre-trained three classes of representative GNNs: GCNs [18], GATs [32], and GraphSAGE [15]. For each dataset, all methods are applied to the same set of GNNs to ensure comparability. Each GNN model is implemented using PyTorch Geometric, with 3 layers and a hidden dimension of 32. We use 70/15/15 split for train/val/test. Each model is trained for 200 epochs using Adam optimizer ( $lr = 0.01$ ).

**Algorithms.** We implemented naiveChase, ApxlChase and HeulChase, compared with two GNN explainers where subgraph structures can be derived for fair comparison. (1) *GNNEExplainer* [35], an established post-hoc method that identifies subgraphs most influential to the model’s prediction by optimizing a continuous edge mask through gradient-based learning. (2) *PGExplainer* [24], a learning-based method that learns a neural edge generator to produce explanations conditioned on node embeddings. Their generated subgraphs are treated as witnesses for comparison. In addition, to test the scalability, we also implemented paralChase, a parallelized implementation of Inf-Chase in a shared memory, but by partitioning the inference query workload (see details in [1]).

**Evaluation Metrics.** We report the efficiency and effectiveness of all the methods. (1) For Efficiency, for online algorithms (e.g., ApxlChase, HeulChase) and naiveChase, we report the total response time including generation, verification, and window maintenance until no changes can be made; for baseline GNN explainers, we report the total runtime needed to generate the explanation subgraphs with the same environment, data, and task configuration. (2) For Effectiveness, we evaluate three metrics: *Coverage*, *Conciseness*, (both are as defined in Section 3), and *Fidelity* [36]. This measure quantifies how well a witness  $G_s$  preserves the model’s output embeddings on  $G$  (“faithfulness”):  $fidelity^-(G_s) = \Pr(Q(M, v_t, G)) - \Pr(Q(M, v_t, G_s))$ , the smaller, the better.

**Environments.** All experiments are conducted on CWRU HPC cluster. Each compute node has an Intel Xeon Silver 4216 CPU at 2.10GHz (16 cores, 32 threads), 125GB of RAM, and a Tesla V100-SXM2 GPU with 32GB of memory. SLURM allocates access to a single node per test. Parallel variants utilize up to 20 CPU threads with GPU acceleration using NVIDIA driver version 560.35, and CUDA version 12.6.

### 7.2 Experimental Results

**Exp-1: Efficiency.** We compare the runtime of all the methods over all datasets and GNN models. For simplicity, method names are abbreviated consistently as follows: APXC (ApxlChase), HEUC (HeulChase), GEX (GNNEExplainer), PGX (PGExplainer), and EXH (exhaustive naiveChase). Unless otherwise specified, we adopt ( $K=6$ ,  $k=8$ ,  $L=2$ ), incompleteness=5%,  $|\Sigma|=20$ ,  $\frac{|V_t|}{|V|}=1\%$  by default. Here, *incompleteness* denotes the proportion of missing edges in  $G$ ,  $|V_t|$  denotes the number of sampled test nodes.

**Overall Efficiency.** Our methods consistently outperform baseline approaches in runtime across all datasets and GNN architectures. As shown in Fig. 8(a), both ApxlChase and HeulChase achieve substantial speedups compared to naiveChase (fixpoint constraint enforcement). For example, ApxlChase (resp. HeulChase) is 10.6 (resp. 554) times faster than naiveChase on Cora, and 35.4 (resp.

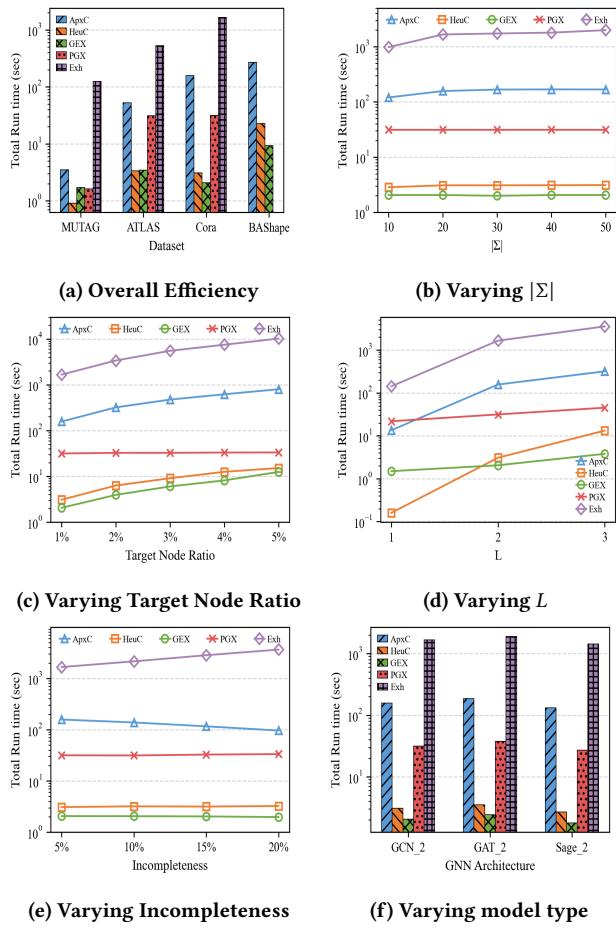


Figure 8: Performance Evaluation: Efficiency

136) times faster on MUTAG, due to their bounded enforcement and optimization strategies. HeuChase is on average 21 times faster than ApxChase, due to its constrained setting on tree-like data and processing. Among baselines, GNNExplainer is the fastest method, as it performs local mutual information optimization without additional training. PGExplainer requires training on the entire graph, leading to higher runtime on large datasets such as Cora, and failing to scale to BAShape.

*Impact of  $|\Sigma|$ .* We fix  $L=2, K=6, k=8$ , Incompleteness = 5%,  $\frac{|V_t|}{|V|} = 1\%$  and vary the number of constraints  $|\Sigma| \in \{10, 20, 30, 40, 50\}$  to evaluate the impact of constraint size. As shown in Fig. 8(b), both ApxChase and HeuChase are not very sensitive. HeuChase is less sensitive than ApxChase due to that the tree constraints and the requirement of spanning tree output makes it less sensitive to the pattern size. GNNExplainer and PGExplainer are insensitive as they do not incorporate constraint checking. naiveChase is the most sensitive, as more constraints incur longer fixpoint computation.

*Varying the size of test set  $|V_t|$ .* We vary the proportion of the target nodes from 1% to 5% on Cora while fixing other factors to their default values. As shown in Fig. 8(c), both ApxChase and HeuChase exhibit an approximately linear growth in total runtime as  $|V_t|$

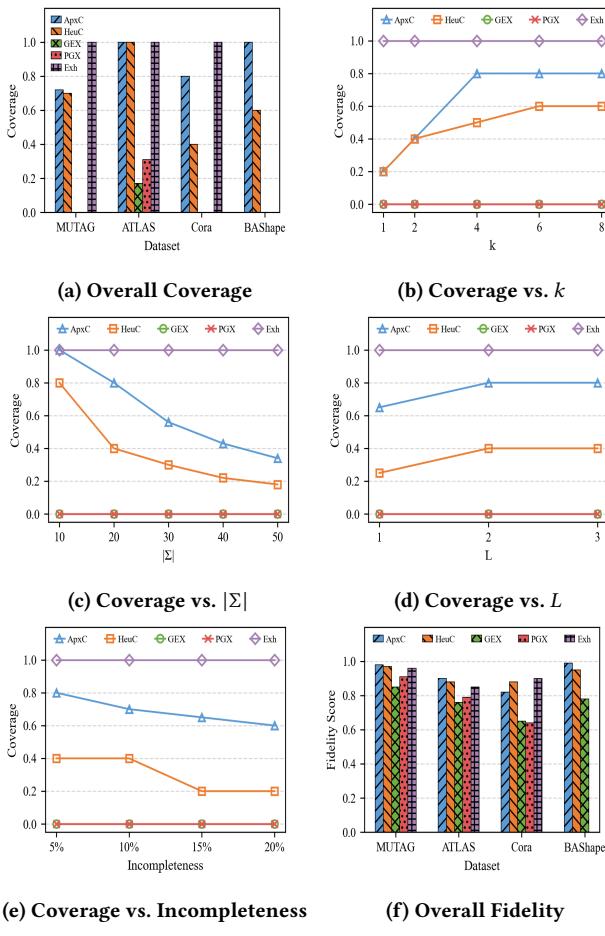
grows. This is expected, since each test node potentially triggers a separate round of verification and constraint-based grounding. Despite the growth, HeuChase remains consistently “lightweight” – under 15 seconds even at 5% sampling – demonstrating its scalability and efficiency in grounding large inference query workload. In contrast, naiveChase incurs a cost that grows rapidly, due to the high cost of fixpoint enforcing process. GNNExplainer is more sensitive compared with PGExplainer as a posthoc approach, as the latter always performs global training over entire graphs.

*Varying GNN layer  $L$ .* We used GCN and varied the number of layers  $L \in \{1, 2, 3\}$ , with all other factors held constant. Fig. 8(d) shows that both ApxChase and HeuChase exhibit sublinear growth in runtime with respect to model depth, demonstrating their scalability with increasing neighbor sizes. In contrast, naiveChase has a significant increase in cost (from 142s to 3,566s) as  $L$  grows, indicating the exponential overhead of constraint enforcement on larger induced subgraphs. GNNExplainer and PGExplainer remain relatively stable, as the former optimizes fixed-size local masks and the latter relies on a pre-trained edge generator independent of  $L$ .

*Varying Incompleteness.* We examine how the total runtime varies as the degree of graph incompleteness increases from 5% to 20%. As shown in Fig. 8(e), ApxChase takes less time as incompleteness increases. This benefits from its candidate generation mechanism, as more edges are missing, less subgraph can be trained in Chasephase, and fast backchase has higher chance to be involved to find  $k$ -gronded witnesses faster. naiveChase, on the other hand, takes more time, and often generates useless or redundant for witness grounding. HeuChase remains largely unaffected, as its arborescence-based generation explores a fixed number of candidate edges regardless of missingness. GNNExplainer and PGExplainer are also less sensitive, as their local optimization and global training processes are independent of structural incompleteness.

*Varying GNN model type.* We examine the influence of backbone GNN architectures on runtime by evaluating all methods on Cora using three representative models: 2-layer GCN, GAT, and GraphSAGE. As shown in Fig. 8(f), our ApxChase and HeuChase exhibit stable performance across different architectures, with runtime difference primarily reflecting the computational complexity of the underlying GNN forward passes. HeuChase remains consistently lightweight (around 3 seconds) across all backbones, demonstrating its robustness and low dependency on model complexity. In contrast, the exhaustive-repair baseline exhibits the largest runtimes across all backbones, with the highest cost on GAT (1,898s). Among all baselines, GNNExplainer remains the fastest, while PGExplainer incurs moderate overhead due to its training process. Overall, we observe a consistent cross-method trend across different GNN backbones, where the relative efficiency ordering of methods remains stable and largely reflects their inherent algorithmic complexity rather than model-specific variations.

**Exp-2: Effectiveness.** We next evaluate the overall quality of witness generation in terms of conciseness, coverage, and fidelity. All the tests are conducted under the default configuration of ( $K=6, k=8, L=2$ , incompleteness=5%,  $|\Sigma|=20, |V_t|/|V|=1\%$ ), unless otherwise specified. Unless otherwise noted, we report the impact of individual factors on the Cora dataset.



**Figure 9: Performance Evaluation: Effectiveness**

**Overall Coverage.** We first compare the coverage over all the datasets under the default configuration, as shown in Fig. 9(a). `naiveChase` attains full coverage as it enforces all constraints before witness generation. Both `ApxlChase` and `HeulChase` consistently achieve high coverage, with `ApxlChase` reaching near-complete grounding on `ATLAS`, `Cora`, and `BAShape`, and `HeulChase` performing comparably except on `Cora` where its tree-like witnesses limit the ability to satisfy cyclic constraints. `GNNExplainer` and `PGExplainer`, which do not consider generating explanations that are grounded by constraints, yield near-zero coverage on most datasets. An exception occurs on `ATLAS`, where the underlying  $L$ -hop neighborhoods are mostly tree-structured, allowing them to be grounded by several constraints with tree patterns.

**Coverage vs. budget  $k$ .** We study how the constraint coverage changes with the budget  $k \in \{1, 2, 4, 6, 8\}$  on `Cora`, while keeping other factors fixed. As shown in Fig. 9(b), the coverage of `ApxlChase` rapidly increases with  $k$  and stabilizes at 0.8, indicating that its flexible edge-insertion process efficiently grounds most constraints even under limited budgets. By contrast, `HeulChase` achieves slightly lower coverage, which can be attributed to its arborescence-based generation: the tree-like witness structure lacks the flexibility to

form cycles, making it inherently less effective for grounding constraints that require closed patterns (e.g., triangle or square rules). Both `GNNExplainer` and `PGExplainer` fail to ground any constraints, as they operate purely on model-centric optimization without enforcing rule consistency. The exhaustive-repair baseline consistently achieves full coverage across all budgets, since all constraints are pre-enforced during its repair phase, serving as the upper bound for constraint satisfaction.

**Coverage v.s.  $|\Sigma|$ .** We vary the size of constraints  $|\Sigma| \in \{10, 20, 30, 40, 50\}$  while fixing other factors at their default values. As shown in Fig. 9(c), both `ApxlChase` and `HeulChase` exhibit a gradual decline in coverage as  $|\Sigma|$  increases. This is expected, since under a fixed budget  $k$ , each method can only ground a limited number of constraints, making it harder to find grounded witnesses with high coverage. On the other hand, the trend is not very sensitive. We found that `ApxlChase` exploits  $G_\Sigma$  and  $M$  well to first reason with constraints that have high overlapping with chased ones, allowing the algorithms to maintain a reasonable coverage even for large  $|\Sigma|$ . `GNNExplainer` and `PGExplainer` remain unaffected, as they do not incorporate constraint grounding.

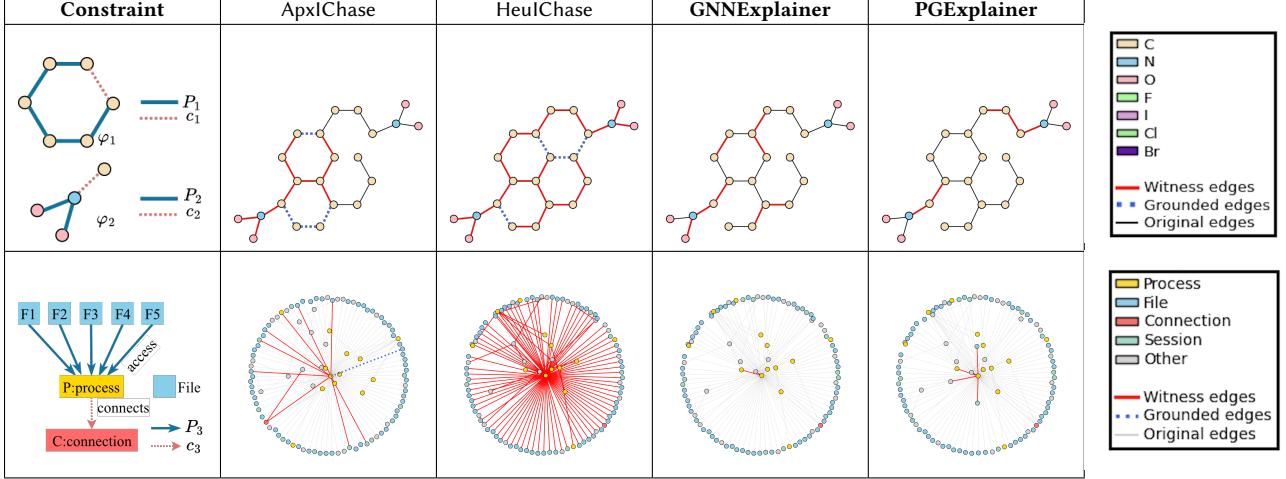
**Coverage vs.  $L$ .** We further analyze the effect of model depth (equivalently, the neighborhood hop count  $L$ ) on constraint coverage, varying  $L \in \{1, 2, 3\}$  while fixing other factors. As shown in Fig. 9(d), both `ApxlChase` and `HeulChase` have coverage improved with larger  $L$ , as a wider receptive field exposes more contextual edges and node dependencies, enabling more constraints to be grounded. The improvement plateaus beyond  $L=2$ , indicating that most applicable constraint patterns are already captured within two hops of the target node. The rest methods remain insensitive due to the same reasoning in our previous tests.

**Coverage vs. Incompleteness.** Varying graph incompleteness from 5% to 20% missing links, we report coverage while keeping other factors fixed. As shown in Fig. 9(e), the coverage of `ApxlChase` and `HeulChase` drops as incompleteness increases. Under a fixed budget, a higher fraction of missing edges reduces the chance to “trigger” the enforcement of constraints, making it harder to ground all applicable constraints. The removal of edges limits the variety of tree-shaped candidates, affecting `HeulChase` to collectively cover more constraints with arborescences.

**Overall Fidelity.** Table 9(f) reports the overall  $1 - \text{fidelity}^-$  scores (the higher, the better) across all datasets under the default configuration. Our `ApxlChase` and `HeulChase` consistently achieve the highest fidelity values on all cases, demonstrating their ability to generate witnesses that are both grounded and preserve model predictions. Notably, the improvement is most pronounced on datasets with more rigid structure schema, connectivity patterns or generation rules, such as `MUTAG` and `BAShape`. Our methods can exploit such constraints that naturally align with data generation, which are “faithful” with the model’s decisions ( $1 - \text{fidelity}^- \approx 0.99$ ).

Interestingly, `ApxlChase` and `HeulChase` sometimes outperform GNN explainers and `naiveChase`. This verifies our analysis that fully enforcing all constraints may not always improve model faithfulness --instead, it may overly enforce new structures and noises, thereby degrading the model’s predictive behavior. These results

**Table 2: Case study: Application of Grounded Witness Generation for Molecular property analysis and Cyberattack detection.**



highlight the interpretive reliability of constraint-grounded witnesses in preserving and faithfully characterizing model behavior.

**Exp-3: Scalability Test.** We evaluate the scalability of our parallel framework (paralChase) under both large-scale real and synthetic graphs, with full algorithmic details provided in the full version [1]. Each experiment adopts  $L=2$ ,  $k=8$ ,  $K=6$ , and 100 target nodes by default. A coordinator induces all  $L$ -hop subgraphs and assigns them to  $p$  processors using load-balanced partitioning by subgraph size. In general, our methods scale well even at graphs of billion-level. We present the detailed analysis in [1].

**Summary.** Overall, ApxlChase and HeulChase strike a balance between computational efficiency and constraint coverage, are able to preserve high faithfulness to graph model output, and scale well over large-scale graphs (at billion-scale).

**Exp-4: Case Studies.** We next showcase how our methods effectively links predictions to real data evidence declared by constraints.

**Case I: Grounding Molecular Property Analysis.** Table. 2(Row 1) illustrates representative cases from the MUTAG dataset. Two constraints are considered on the left: (1)  $P_1 \rightarrow c_1$  enforces that a chain of six carbon atoms should form a benzene-like cycle, and (2)  $P_2 \rightarrow c_2$  encodes an oxygen–nitrogen bonding pattern, reflecting a common functional linkage in mutagenic compounds. Starting from an (incomplete) molecular graph where several bonds missing, ApxlChase detects a witness (red edges) that is grounded by the second constraint (oxygen–nitrogen linkage), and under a budget of  $k=4$ , it can also be linked, by 4 grounded edges (blue dotted) to two benzene rings, suggesting a supporting aromatic structures that is chemically valid. HeulChase similarly constructs a spanning-tree–like witness that is grounded by the same functional linkage rule and enforces ring-closure constraint within the available budget. In comparison, both GNNEExplainer and PGExplainer produce fragmented subgraphs that lack chemical interpretability, and fail to be grounded by any of the given constraints.

**Case II: Data Exfiltration Attack Investigation.** We analyze a backdoor process flagged by a 3-layer GraphSAGE model for credential theft and data exfiltration. Table 2 (Row 2) compares reconstructed attacks under constraint  $\varphi_3 : P_3 \rightarrow c_3$ , requiring that any process accessing five sensitive files—F1:/etc/passwd, F2:.ssh/id\_rsa, F3:cookies.db, F4:confidential.pdf, F5:credentials.txt—must also show outbound connection C:198.51.100.25:443. (1) ApxlChase yields a clear star-shaped witness centered on the backdoor, linking it to all five files and C. Orange grounded edges confirm satisfaction of  $P_3 \rightarrow c_3$ , capturing both phases of the attack: systematic credential collection (backdoor  $\rightarrow$  F1–F5) and exfiltration (backdoor  $\rightarrow$  C). The result provides investigators with a complete, constraint-aligned explanation of theft and transmission. (2) HeulChase also recovers all six critical nodes but embeds them in a dense arborescence that mixes some unrelated files and processes. This may be due to its edge construction based on similarity. Yet as  $\varphi_3$  holds, it can suggest structures to the downstream analysis to further isolate the true attack paths. (3) GNNEExplainer detects edges to only F1 and F4, omitting F2–F3–F5 and C; the missing consequent and partial antecedent violate  $\varphi_3$ , leaving an incomplete narrative that suggests benign file access. (4) PGExplainer approximates partial coverage but deviates from the constraint with connections to auxiliary sessions, missing critical files (F2, F3, F5). Its probabilistic sampling breaks the antecedent–consequent structure, yielding ungrounded witnesses.

## 8 CONCLUSION

We have introduced  $k$ -grounded witnesses to characterize subgraphs that are necessary to reconstruct graph inference query results, and meanwhile be grounded by graph data constraints. We introduced quality measures for  $k$ -grounded witnesses, and verified the hardness of computing optimal  $k$ -grounded witnesses. We present both a principled algorithm based on a variant of Chase and Backchase process, and two practical specifications to efficiently generate  $k$ -grounded witnesses as general subgraphs and trees. We have experimentally verified that our algorithms outperform existing graph explanation techniques on generating grounded witnesses, and their applications in drug design and cybersecurity.

## REFERENCES

- [1] 2025. Full version. [https://github.com/anthonyche/Grounded\\_Witness/blob/main/Grounding\\_Graph\\_Inference\\_Analysis\\_with\\_Data\\_Constraints.pdf](https://github.com/anthonyche/Grounded_Witness/blob/main/Grounding_Graph_Inference_Analysis_with_Data_Constraints.pdf)
- [2] Abdulkadir Alsaheel, Yuhong Nan, Shiqing Ma, Le Yu, Gregory Walkup, Z Berkay Celik, Xiangyu Zhang, and Dongyan Xu. 2021. {ATLAS}: A sequence-based learning approach for attack investigation. In *30th USENIX Security Symposium (USENIX Security 21)*. 3005–3022.
- [3] Burouj Armgaaan, Manthan Dalmia, Sourav Medya, and Sayan Ranu. 2024. Graph-Trail: Translating GNN predictions into human-interpretable logical rules. *Advances in Neural Information Processing Systems 37* (2024), 123443–123470.
- [4] Michael Benedikt, George Konstantinidis, Giansalvatore Mecca, Boris Motik, Paolo Papotti, Donatello Santoro, and Eftymia Tsamoura. 2017. Benchmarking the chase. In *PODS*. 37–52.
- [5] Peter Buneman, Sanjeev Khanna, and Tan Wang-Chiew. 2001. Why and where: A characterization of data provenance. Springer, 316–330.
- [6] Marco Calautti, Mostafa Milani, and Andreas Pieris. 2023. Semi-Oblivious Chase Termination for Linear Existential Rules: An Experimental Study. *Proc. VLDB Endow.* 16, 11 (2023), 2858–2870.
- [7] Ming Chen, Zhewei Wei, Bolin Ding, Yaliang Li, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. 2020. Scalable graph neural networks via bidirectional propagation. *Advances in neural information processing systems 33* (2020), 14556–14566.
- [8] James Cheney, Laura Chiticariu, Wang-Chiew Tan, et al. 2009. Provenance in databases: Why, how, and where. *Foundations and Trends® in Databases* (2009).
- [9] Asim Kumar Debnath, Rosa L Lopez de Compadre, Gargi Debnath, Alan J Shusterman, and Corwin Hansch. 1991. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry* 34, 2 (1991), 786–797.
- [10] Alin Deutsch, Lucian Popa, and Val Tannen. 2001. Chase & backchase: A method for query optimization with materialized views and integrity constraints. (2001).
- [11] Jinlong Du, Senzhang Wang, Hao Miao, and Jiaqiang Zhang. 2021. Multi-Channel Pooling Graph Neural Networks. In *IJCAI*. 1442–1448.
- [12] Jack Edmonds et al. 1967. Optimum branchings. *Journal of Research of the national Bureau of Standards B* 71, 4 (1967), 233–240.
- [13] Wenfei Fan, Xin Wang, Yinghui Wu, and Jingbo Xu. 2015. Association rules with graph patterns. *Proceedings of the VLDB Endowment (PVLDB)* 8, 12 (2015), 1502–1513.
- [14] Wenfei Fan, Yinghui Wu, and Jingbo Xu. 2016. Functional dependencies for graphs. In *Proceedings of the 2016 international conference on management of data*. 1843–1857.
- [15] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *NeurIPS* (2017).
- [16] Yuxing Han, Ziniu Wu, Peizhi Wu, Rong Zhu, Jingyi Yang, Liang Wei Tan, Kai Zeng, Gao Cong, Yanzhao Qin, Andreas Pfadler, Zhengping Qian, Jingren Zhou, Jiangeng Li, and Bin Cui. 2022. Cardinality Estimation in DBMS: A Comprehensive Benchmark Evaluation. *Proceedings of the VLDB Endowment* 15, 4 (2022), 752–765. <https://doi.org/10.14778/3503585.3503586>
- [17] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. *NeurIPS* (2020).
- [18] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).
- [19] Douglas J Klein. 2010. Centrality measure in graphs. *Journal of mathematical chemistry* 47, 4 (2010), 1209–1223.
- [20] Ryoji Kubo and Djallel Difallah. 2025. RAW-Explainer: Post-hoc Explanations of Graph Neural Networks on Knowledge Graphs. *arXiv preprint arXiv:2506.12558* (2025).
- [21] Wanyu Lin, Hao Lan, and Baochun Li. 2021. Generative causal explanations for graph neural networks. In *International conference on machine learning*. PMLR, 6666–6679.
- [22] Wanyu Lin, Hao Lan, Hao Wang, and Baochun Li. 2022. Orphicx: A causality-inspired latent variable model for interpreting graph neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 13729–13738.
- [23] Ana Lucic, Maartje A Ter Hoeve, Gabriele Tolomei, Maarten De Rijke, and Fabrizio Silvestri. 2022. Cf-gnnexplainer: Counterfactual explanations for graph neural networks. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 4499–4511.
- [24] Dongsheng Luo, Wei Cheng, Dongkuan Xu, Wenchao Yu, Bo Zong, Haifeng Chen, and Xiang Zhang. 2020. Parameterized explainer for graph neural network. *Advances in neural information processing systems 33* (2020), 19620–19631.
- [25] Michael J Maher and Divesh Srivastava. 1996. Chasing constrained tuple-generating dependencies. In *Proceedings of the fifteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*. 128–138.
- [26] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. 2000. Automating the construction of internet portals with machine learning. *Information Retrieval* (2000).
- [27] Paolo Pareti, George Konstantinidis, Timothy J Norman, and Murat Sensoy. 2019. SHACL constraints with inference rules. In *International Semantic Web Conference*. Springer, 539–557.
- [28] Marcus Schaefer and Christopher Umans. 2002. Completeness in the polynomial-time hierarchy: A compendium. *SIGACT news* 33, 3 (2002), 32–49.
- [29] Yufan Sheng, Xin Cao, Kaiqi Zhao, Yixiang Fang, Jianzhong Qi, Wenjie Zhang, and Christian S. Jensen. 2025. ACE: A Cardinality Estimator for Set-Valued Queries. *Proceedings of the VLDB Endowment* 18, 7 (2025), 2112–2125. <https://doi.org/10.14778/3734839.3734848>
- [30] Larissa C Shimomura, George Fletcher, and Nikolay Yakovets. 2020. GGDs: graph generating dependencies. In *CIKM*.
- [31] Qi Song, Mohammad Hosseini Namaki, Peng Lin, and Yinghui Wu. 2020. Answering why-questions for subgraph queries. *IEEE Transactions on Knowledge and Data Engineering* 34, 10 (2020), 4636–4649.
- [32] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, Yoshua Bengio, et al. 2017. Graph attention networks. *stat* (2017).
- [33] Minh Vu and My T Thai. 2020. Pgmn-explainer: Probabilistic graphical model explanations for graph neural networks. *Advances in neural information processing systems 33* (2020), 12225–12235.
- [34] Xiang Wang, Yingxin Wu, An Zhang, Fuli Feng, Xiangnan He, and Tat-Seng Chua. 2022. Reinforced causal explainer for graph neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2022), 2297–2309.
- [35] Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. 2019. Gnnexplainer: Generating explanations for graph neural networks. *Advances in neural information processing systems 32* (2019).
- [36] Hao Yuan, Haiyang Yu, Jie Wang, Kang Li, and Shuiwang Ji. 2021. On explainability of graph neural networks via subgraph explorations. In *International conference on machine learning*. PMLR, 12241–12252.
- [37] Hongkuan Zhou, Ajitesh Srivastava, Hanqing Zeng, Rajgopal Kannan, and Viktor Prasanna. 2021. Accelerating large scale real-time GNN inference using channel pruning. *arXiv preprint arXiv:2105.04528* (2021).
- [38] Mingzhe Zhu, Liwang Zhu, Huan Li, Wei Li, and Zhongzhi Zhang. 2024. Resistance distances in directed graphs: Definitions, properties, and applications. *Theoretical Computer Science* 1009 (2024), 114700.
- [39] Chuanyu Zong, Zefang Dong, Xiaochun Yang, Bin Wang, Tao Qiu, and Huaijie Zhu. 2023. Efficiently answering why-not questions on radius-bounded k-core searches. In *International Conference on Database Systems for Advanced Applications*. Springer, 93–109.
- [40] Chuanyu Zong and Chengwei Zhang. 2024. Effectively answering why questions on structural graph clustering. *Applied Soft Computing* 154 (2024), 111405.