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 URL_TO_YOUR_ARTIFACTS.

1 INTRODUCTION

Graph machine learning (ML) models, such as graph neural networks (GNNs), have shown promise results in graph analytical 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 $Q(M, V_T, G)$.

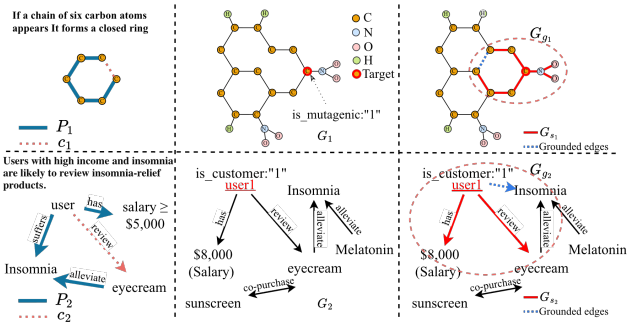


Figure 1: Motivation Example

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: Considering two scenarios as shown in Fig. 1, (1) Chemists heavily rely on empirical rules (facts) for molecular analysis tasks, e.g., carbon ring detection. φ_1 specifies a rule: *"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 GNN as a *mutagenic atom-i.e.,* an atom that contributes to the molecule's mutagenicity. This is due to, even there are missing edges in $G1$, two benzene-like structures that should have formed stable 6-atom rings are instead presented as a 10-atom ring, the model identifies it and provides the provenance with a limited explanation of why the model identifies the structure as

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

"mutagenicity" with unconvincing evidence for the chemist's interpretation. (2) A similar issue arises in user behavior recommender systems. Considering a common customer behavior pattern φ_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 the eye cream. A business analyst may ask: Why is this user classified as a potential buyer? Ideally, the provenance should connect this decision to the user's insomnia condition and the product's therapeutic relation (*alleviates insomnia*). Due to the missing information indicating the user's insomnia, the provenance solely relies on historical correlations between eye cream and high income, while ignoring the health condition that actually drives the purchase. In this case, the recommendation model may misinterpret such correlations and even suggest unrelated products (*e.g.*,, sunscreen), yielding a misleading justification that deviates from the true behavioral pattern. Together, these two cases show that graph incompleteness may introduce unconvincing provenance of graph inference, concealing the true relational dependencies behind model predictions. This motivates constraint-grounded provenance, which restores or enforces missing links to recover faithful reasoning paths. □

Example 2: Following example 1, a more desired provenance structure should not only respect to the model's prediction but also capture comprehensive semantical relations. In the molecular scenario, as shown in Fig. 1, a 3-layer GCN M that classifies the highlighted atom in G as "mutagenic". The red edges form a subgraph $G_{s_1} \subseteq G$, which supports the model output-*i.e.,* $Q(M, v_t, G) = Q(M, v_t, G_{s_1})$. The area enclosed by the pink dotted oval represents G_{g_1}, the grounded provenance extended from G_{s_1}. By enforcing $P_1 \rightarrow c_1$, a grounded edge(blue dotted) is added to close the 6-atom ring, thus the benzene ring is reconstructed. In practice, such grounding preserves the model's prediction while aligning the provenance with chemical domain knowledge. In the recommender system scenario, (bottom of Fig. 1), a 3-layer GAT M', that predicts target node user1(v'_t) as a potential buyer of the eye cream, the red edges forms G_{s_2} s.t. $Q(M', v'_t, G_2) = Q(M', v'_t, G_{s_2})$, the pink dotted oval represents the grounded provenance G_{g_2}, obtained by enforcing $P_2 \rightarrow c_2$-that high-income (over 5000\$) users with insomnia tend to review or purchase insomnia-relief products. The grounded edge(blue dotted) restores the missing link(user1 $\rightarrow$ Insomnia), yielding a more complete and causally aligned rationale that helps the recommender to produce more relevant suggestions(*e.g.,* Melatonin) rather than unrelated products such as sunscreen. □

This calls for cost-effective methods to generate provenance data for graph inference queries. Likening data provenance that compute provenance data as influential fraction of the input data for structured queries [5], **provenance for graph inference** seeks to trace the rationale behind a graph model's decision by computing subgraphs that are "faithful" to the model output, as the query result of an inference query. Such provenance *should* capture not only the model's internal decision-making logic but also be contextualized by external knowledge and constraints that reflect the user's interpretation needs.

We are interested in developing a Why-provenance mechanism for Inference queries over GNNs. Moreover, the generated structure must conform to a set of external data constraints, hence making it "grounded" by real-world settings or the user's knowledge level, to make such provenance useful and usable for the in-context information needs. 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 Σ, 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' may *not* necessarily be a subgraph of G.

Related work. We summarize related work below.

Why provenance. 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]. Provenance plays a key role in ensuring transparency, trust, and accountability in data-driven systems. A central theme in this line of research is the formulation of specific **why-questions**, such as: "Why is a particular tuple present in the output?", "Why is an expected tuple missing?", or "Why is one output ranked above another?" These have been formalized respectively as **why-provenance** [5], **why-not provenance** [30], and **why-so / why-rank provenance** [19, 39]. To operationalize such explanations, semiring-based models [16] provide a compositional framework for annotating and combining provenance information through algebraic expressions. Other approaches model provenance using derivation trees or traces to capture the data transformation paths [15]. In addition, causality-based formulations characterize explanations as minimal changes to inputs that alter the output [29]. These foundational approaches have primarily focused on symbolic query evaluation over relational or semi-structured data. Recent research has extended provenance-style explanations to graph-structured queries, especially for understanding missing or surprising results in subgraph pattern search.

Provenance for graph queries. Recent work has extended provenance-style explanations to graph queries, particularly in scenarios involving subgraph search and graph analytics. [35] makes a first attempt to 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. [45] 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. [44] 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 why-provenance for graph inference, aiming to explain model predictions under structural and semantic 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 [26, 40]), surrogate-based (e.g., PGMExplainer [37]), and causality-inspired approaches (e.g., GEM [23], RCExplainer [38], OrphicX [24]). 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.

Recent work has attempted to improve interpretability through more structured or symbolic reasoning. For example, SubgraphX[41] uses Shapley value-guided Monte Carlo search to extract compact subgraphs that preserve model predictions, while GraphTrail[2] learns common subgraph motifs and encodes them as logic rules for global explanation. RAW-Explainer [21] explores perturbation-based explanations in knowledge graphs but is limited to link prediction and lacks structural alignment. Compared to these methods, our method 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 structural constraints into post-hoc GNN explanation.

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, 20] 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.

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

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. $\widetilde{\mathbf{A}}$ 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"). σ is an activation function.

The *inference process* of a GNN $\mathcal{M}$ with L layers takes as input a graph $G = (X, \widetilde{\mathbf{A}})$, 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 $\mathcal{M}(v, G)$, for the output of a GNN $\mathcal{M}$ at a node $v \in V$.

Deterministic Inference. A GNN $\mathcal{M}$ 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 deterministic, 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 $\mathcal{Q} = (M, V_T, G)$ 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 $\mathcal{Q}(M, V_T, G_1)$ 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 "constraint"). 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 $<u_s, u_p, u_o>$. 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 op a, where a is a constant, and 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.

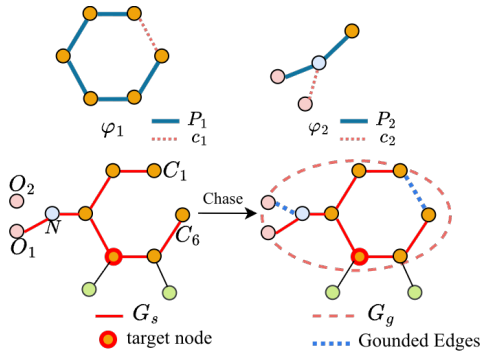


Figure 2: Graph data constraint and witness grounding. Two structural constraints are illustrated: $\varphi_1 = (P_1, c_1)$ closes a carbon ring, and $\varphi_2 = (P_2, c_2)$ completes a missing bond between nitrogen and oxygen. During Chase, a subgraph G_s matching P_1 and P_2 (red edges) is identified, 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.

(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 φ, we say a graph G *satisfies* φ 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 Σ, the graph G satisfies Σ, denoted as $G \models \Sigma$, if for every constraint $\varphi \in \Sigma$, $G \models \varphi$.

We say a graph data constraint φ is *trivial*, if $c \in P$. Given a pair of graph data constraints (φ, φ'), we say $\varphi = (P, c)$ implies $\varphi' = (P', c)$, if $P \subseteq P'$. In our work, we assume a set of graph data constraints Σ 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 [34], or SHACL [31].

Enforcement. Chase [3, 6] has been a powerful tool for query expressiveness, constraint reasoning and enforcement. Given a set of data constraints Σ and a dataset D, a Chase process is a general technique that can transform D such that $D \models \Sigma$. When Σ is used to express a query Q (hence a set of query constraints), Chase can be used for evaluating conjunctive queries. If Σ is defined as data quality rules or constraints [27], Chase enforces Σ as a data repairing process. Applying Chase twice yields Chase&BackChase, a technique used for query optimization or query suggestion [10].

Example 4: Let $\varphi_1 = (P_1, c_1)$ 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 the 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, producing a graph $G_g \models \{\varphi_1, \varphi_2\}$ that satisfies both constraints and suggests a chemically consistent molecular structure. □

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 over incomplete graphs, 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: $G \rightarrow \mathbf{H}_v^{(L)}$, 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. In this case, 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 $\mathbf{H}_{v_t}^{(3)}$, , while removing any of these red edges would change the predicted label, (*i.e.,* is_mutagenic:"0"). □

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 [40] or a counterfactual witness [25], or be implicitly induced from a learned masked adjacency matrix [26].

3 GROUNDING WITNESSES WITH CONSTRAINTS

A witness alone may not be linked to meaningful real-world structures ("ungrounded"), due to that they may be only faithful to the models [40] which are "misaligned" to reflect the context of the downstream analytical tasks, or the missing links in incomplete graphs [4], 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 φ, 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 φ, 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 ΔE that are not in G, such that (1) G_g contains ΔE and G_s, and (2) $G_g \models \varphi$ ($|\Delta E| \leq k$).

We remark that the symbol $\oplus$ means "extended" but not "union", *i.e.,* G_g may contain G_s, a set of new edges ΔE 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-grouned by a set of data constraints Σ, if for every constraint φ, $G_s \models_k \varphi$. 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) ΔE 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 φ_1 enforces the carbon ring closure and φ_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 ΔE 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 Σ. □

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 φ. The constraints φ, 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 ΔE readily suggest "how" G should be modified to include missing data, that are specifically relevant to decision making process of M. This also indicate useful training and testing triples for GNNs. (3) Adversarial graph learning. The ΔE 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 φ, 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 $G^L_{v_t}$, the L-hop subgraph of a test node v_t. The conciseness of G_s is defined as $\mathsf{conc}(G_s) = 1 - \frac{|G_s|}{|G^L(v_t)|}$. 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 φ - that is, $G_s \models \varphi$. Given φ, 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 ΔE required to have G_s k-bounded by φ, denoted as an alignment penalty. We introduce a function $\mathsf{aln}(G_s, \varphi) = 1 - \frac{|\Delta E|}{k}$. Higher score indicates smaller alignment penalty. Here $|\Delta E| \in [0, k]$. A 0-grounded G_s has $\mathsf{aln}(G_s, \varphi) = 1$.

Consistency. Given a set of constraints Σ, a witness G_s ideally is k-grounded by all $\varphi \in \Sigma$. We define the consistency measure of G_s, denoted as con, as $\mathsf{con}(G_s) = \frac{|\Sigma(G_s, k)|}{|\Sigma|}$, where $\Sigma(G_s, k) = \{\varphi | G_s \models_k \varphi, \varphi \in \Sigma\}$, *i.e.,* the subset of Σ 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 $F(G_s)$:

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

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 conc, aln and con are scaled by some constant. This ensures the invariance of G^*_s regardless of how the user preference (α, β, γ) 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 Σ, 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 cite full version).

Theorem 1: *The witness generation problem is Π^P_2-hard for an inference query $Q(M, v_t, G)$ with fixed G and M.* □

Proof sketch: We construct a polynomial-time reduction from the min-max Clique problem [33], 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 Σ 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, 43]. If there is a witness that satisfies Σ, 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 Π^P_2-complete, the hardness follows. □

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, Σ, and a local budget k, naiveChase first enforces Σ 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 φ 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 Σ, (1)* I-Chase *always terminates after a finite steps of enforcement, and (2) The computation has the Church-Rosser Property.* □

(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 cite full version), it remains infeasible in practice for large G with missing edges. (1) The enforcement of Σ 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 $P \cup c$.

We next introduce efficient algorithms, to generate k-grounded witnesses without an exhaustive enforcement of Σ 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 Σ 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 Σ, to make each witness $\mathcal{G}_s$ k-grounded by Σ. 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 Σ_b is an inverse counterpart of some $\varphi = (P, c)$ in Σ.

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.

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

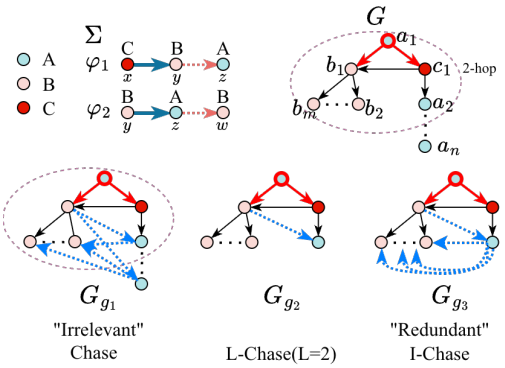


Figure 3: Generating grounded witnesses (Inf-Chase); unlike I-Chase as a fixpoint computation, L-Chase reasons with Σ up to L times from a set of "source" constraints Σ_s.Fig3 is not showing a Inf-Chase process

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 Σ_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 ΔE 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 φ^{-1}. Including a connected subgraph G_g with G_c and ΔE ensures to generate a set of minimal witness $\mathcal{G}_g$. This step introduces at most $|\Sigma|k$ new edges.

Example 7: Give an example to show the running of Inf-Chase, at high level. And compare I-Chase and L-Chase. Consider the constraint set $\Sigma = \{\varphi_1, \varphi_2\}$ shown in Fig. 3, where $\varphi_1 : (P_1, c_1)$ enforces that a node of type C connected to a node of type B (*i.e.,* $C \rightarrow B$) should also induce a link from B to a node of type A (*i.e.,* $B \rightarrow A$), and $\varphi_2 : (P_2, c_2)$ enforces that whenever $B \rightarrow A$ holds, a reciprocal link $A \rightarrow B'$ should also exist. The right-hand side of the figure shows a small-world graph G, where the red bold edges denote the subgraph G_s covering the one-hop neighborhood of the target node v_t, and the dashed oval marks its 2-hop receptive field given $L = 2$. The nodes $b_2 \ldots b_m$ form a wide leaf set, while $a_2 \ldots a_n$ extend into a deep chain of descendants. Under Chase, once P_1 is matched, the algorithm exhaustively enforces c_1 across all matches—creating spurious edges such as $b_2 \rightarrow a_2, \ldots, a_n$. Most of ΔE fall outside the receptive field of v_t and thus become irrelevant (see G_{g_1}). In contrast, I-Chase restricts reasoning within $G^L(v_t)$, but in the second round it still redundantly enforces multiple semantically-equivalent edges (*e.g.,* $a_2 \rightarrow b_2, \ldots, b_m$), leading to an over-grounded graph

G_{g_3}. L-Chase provides a balanced alternative: by performing up to L bounded rounds of reasoning within $G^L(v_t)$, it suffices to chase $(L - \text{hop}(G_s))$ rounds to produce an informative and locally grounded result (G_{g_2}) without irrelevant or redundant edges. □

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 φ. For any φ' with $P' \in c$, its inverse form $\varphi'^{-1} = (c, P')$. The backchase phase I-Chase (φ'^{-1}, G_s, k) makes G_s k-grounded, by identify all applicable φ'^{-1} over G_s and enforcing (c, P') over all witnesses $G_s \in \mathcal{G}_s$, treating φ'^{-1} as a tuple("edge")-generation constraint on triples of $G^L(v_t)$. By definition, each G_b derived succesfully ensures to contain G_s as a k-grounded witness for $\varphi' \in \Sigma$. That is, G_g = I-Chase (φ'^{-1}, G_s, k) = I-Chase $(\varphi'^{-1}$,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.

Time cost. Both Chaseand backchase phases takes $|\Sigma|$ calls of I-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 Σ, where p is the size of the largest pattern P a data constraint has in Σ. The inference cost is in $O(L|G^L(v_t)||N^L(v_t)|)$ time [7, 43], 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 ApxIChase and illustrated in Fig. 11, 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 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.* □

Memoization Structures. ApxIChase 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_Σ is a graph pattern obtained as the union of the patterns $P \cup c$, encoding all the triple patterns from data constraints of Σ. Each edge $e = (u_s, u_p, u_o)$ in G_Σ carries a set of labels $\mathcal{L}(e) = \{i | \text{if } e \text{ is in } P_i \text{ or } c_i\}$ for any

Algorithm 1 : ApxIChase

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

```
construct G_Σ:= ⋃_{φ∈Σ} P; initializes M; set G_g:=∅, G_c:=∅,
W_k:=∅; Σ_s:=∅; Σ_b:=∅;
for ℓ = 1 to L do
    Σ_s:= {φ|u.l = v_t.l, u is from φ.P; φ ∈ Σ};
/* Chase phase */
    G_s := G_s ∪ L-Chase (Σ, Σ_s, G^L(v_t), G_Σ, M, 0);
/* initializes backchase */
    set Σ_b := ∅;
    for each G_s ∈ G_s and each φ ∈ Σ do
        if G_s ⊨ φ then Σ_b := Σ_b ∪ φ^{-1};
/* backchase phase */
        G_g := G_g ∪ L-Backchase (Σ, Σ_b, G_s, G_Σ, M, k);
/* maintain current Top-K grounded witness set*/
        W_K := UpdateWK (W_K, G_g, K, B);
return W_K;
```

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

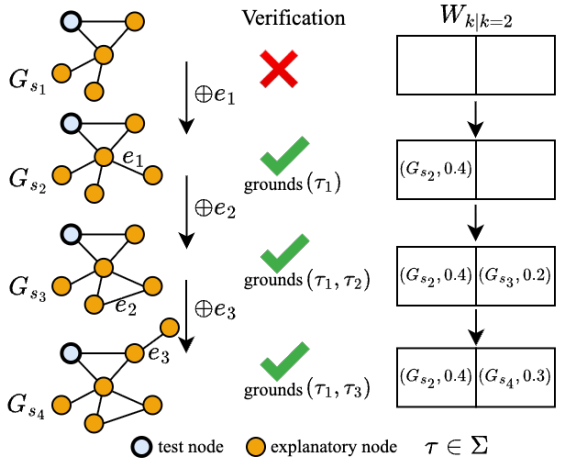


Figure 5: ApxIChase**: a "top-down"** Inf-Chase **simulation.Each candidate subgraph G_{s_i} is generated by incremental edge insertion and verified for witness preservation. If G_{s_i} grounds new constraints $\varphi \in \Sigma$ within budget B, it contributes positive marginal gain $\Delta F(G_s)$ and is admitted into the streaming window W_k. The window (here $k = 2$) stores entries $(G_s, \Delta F(G_s))$ and is updated greedily, ensuring that at most k candidates with the highest marginal contributions are retained.inconsistency**

$\varphi_i = (P_i, c_i)$, *i.e.*, e indicates a common triple pattern in multiple constraints. (2) A shared map $\mathcal{M}$ with size $|\varphi| \times (p + 2)$, with p the size of the largest pattern P from Σ, and for each entry $\mathcal{M}[i]$,

- $\mathcal{M}[i].\mathcal{P}$ is a size-$p + 1$ binary vector, such that each slot $\mathcal{M}[i].\mathcal{P}[j]$ ($j \in [1, p]$) = 1 (resp. $\mathcal{M}[i].\mathcal{P}[p + 1] = 1$) if and only if φ_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 φ_i.

In addition, ApxIChase 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.

Algorithms. The main driving algorithm ApxIChase, and the procedures L-Chase and L-Backchase are illustrated in Figs. 11, 6 and 7, respectively. ApxIChase 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 [?] cite full).

The correctness and time cost follows from its counterpart of Inf-Chase. We present the details in [?] cite full.

Optimization. Algorithm ApxIChase 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 getNext (line 3), which follows a prioritized traversal of G_Σ to prefer φ with pattern edges that has most occurrences over Σ, and grows candidate subgraphs G_c with matching edges accordingly. This favors φ with common evidences, which are typically more credible than those rare counterparts. Other query selectivity or cardinality estimation techniques *e.g.,* [] cite 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.

(3) Procedure L-Backchase reduces enforcement cost by monitoring $\mathcal{M}$ to optimize the enforcement cost. It maintains a lower bound of $|\Delta|$, 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 Σ_s and Σ_b by checking if $\mathcal{M}[i].P[d+1]$ (the consequent) has a match.

(4) ApxIChase also copes with potential large Σ by removing any constraints that can be already logically implied by others, with a set of pruning rules. We present the details in [?]cite full.

These strategies are effective: they improve naive implementation on average **TBF** times, as verified by our experimental study.

Example 8: Figure 5 illustrates the execution of ApxIChase on a 2-hop neighborhood H around target node v_t, where three edges $\{e_1, e_2, e_3\}$ are incrementally inserted to form candidate subrgaphs $\{G_{s_1}, G_{s_2}, G_{s_3}, G_{s_4}\}$ under constraint set $\Sigma = \{\varphi_1, \varphi_2, \varphi_3\}$. Starting from the singleton state $s_0 = (\{v_t\}, \emptyset)$, each edge insertion $\oplus e_i$

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 := 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_Σ, $\mathcal{M}$;
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 6: 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 ΔE 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 7: Procedure L-Backchase

expands the current subgraph and triggers an inference query as a verification step to ensure that the prediction of v_t is preserved. If a candidate G_{s_i} grounds one or more previously uncovered constraints(*e.g.,* φ_1 or φ_1, φ_2) within budget B, it contributes positive marginal gain $\Delta F(G_{s_i})$ and is admitted into the streaming window W_k. As shown for $k = 2$, the window gradually evolves from empty to maintain the top-2 candidates $(G_{s_2}, 0.4)$, $(G_{s_4}, 0.3)$ that achieve the highest marginal contributions, whose grounded constraints jointly form $\Sigma^* = \{\varphi_1, \varphi_3\}$. This demonstrates how ApxIChase greedily constructs compact explanatory subgraphs that maximize overall constraint coverage.to change along with Fig.5 □

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 [28]. 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 Σ has a tree pattern $P \cup c$, (2) the k-grounded witnesses $\mathcal{G}_s$ 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 explanability analysis [], such as embedding similarity, where $w(e) = \text{sim}(h_u^{(L)}, h_w^{(L)}) + \varepsilon_e$, where

$\text{sim}(a,b) = \frac{a^\top b}{\|a\|\cdot\|b\|}$, and ε_e is a small random noise (*e.g.*, Gumbel or Gaussian) that introduces diversity across m samples of trees. Other options include resistant distance []cite, Shapley value [41], or sensitivity [].cite

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. On the other hand, we introduce a fast heuristic, denoted as HeuIChase, to generate G_g that ground witnesses as maximum weighted arborescences.

Algorithm Outline. The algorithm follows the framework IncChase with a L-Chase and a L-Backchase step to generate grounded witnesses. Unlike ApxIChase, HeuIChase 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 arborescence. L-Backchase then applies Edmonds' algorithm [12] to derive G_g over the shrinked network.

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

Analysis. The algorithm HeuIChase ensures the correct generation of provenance structures as k-grounded tree witnesses, and weighted arborescences, following the correctness analysis of IncChase and the correctness of Edmonds' algorithm. It incurs the same worst-case delay time cost as in ApxIChase. 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 Σ) 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 []. The total time cost of HeuIChase 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

We conduct experiments to evaluate the effectiveness, quality, and scalability of our proposed Why-Inf framework for constraint-guided graph explanation.

Datasets. We use six real-world graphs and one synthetic graph for evaluation shown in Table 1.

Dataset	|V|	|E|	# node types	# attributes
BAShape [40]	2,020,000	12,055,704	4	1
Cora [28]	2708	5429	7	1433
ATLAS [1]	59,148	40,823	2	64
MUTAG [9]	17.9(avg.)	19.8(avg.)	7	7
ogbn-Papers100M [18]	111M	1.6B	40	128
Tree-Cycles [40]	67M	1.4B	5	5

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

MUTAG. MUTAG is a molecular graph classification dataset containing 188 chemical compound graphs with an average of 17.9 atoms (nodes) and 19.8 chemical bonds (edges) per molecule. Each graph is labeled by mutagenicity. Nodes represent atoms with 7-dimensional one-hot features encoding atom types (C, N, O, F, I, Cl, Br).

ATLAS. ATLAS [1] is a cybersecurity event graph dataset for intrusion detection. Nodes represent system entities (e.g., process, file, connection), and edges denote causal relations. The dataset contains approximately 59,148 nodes and 40,823 edges with node labels indicating normal or malicious behavior.

Cora. Cora is a citation network dataset with 2,708 papers (nodes) classified into 7 categories [28]. Each node has a 1,433-dimensional feature vector, and edges represent citation links between papers.

BAShape. BAShape is a synthetic dataset generated by attaching house-shaped motifs to a Barabási–Albert base graph [40]. It contains approximately 2.02 million nodes and 12.06 million edges, where node labels correspond to structural roles (top, middle, bottom, or background).

ogbn-Papers100M. OGBN-Papers100M [18] is a large-scale citation network with 111 million papers (nodes) and 1.6 billion citations (edges). Each paper has a 128-dimensional feature vector and is classified into one of 40 research fields.

Tree-Cycles. Tree-Cycles [40] is a synthetic dataset with hierarchical graphs combining tree structures and cyclic connections. It contains approximately 67 million nodes and 1.4 billion edges, with nodes assigned to five categories represented as 5-dimensional one-hot features.

Environments. All experiments are conducted on CWRU-managed HPC cluster. Each compute node used in our evaluation is equipped with 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. We use SLURM to allocate exclusive access to a single node per experiment. Parallel variants (e.g., paraIChase) utilize up to 20 CPU threads and GPU acceleration when available. The CUDA version is 12.6, and the NVIDIA driver version is 560.35.

Graph Models. We have pre-trained three classes of representative GNNs: GCNs [20], GATs [36], and GraphSAGE [17]. For each dataset, all methods are applied to the same set of GNNs to ensure comparability.

Model Architecture. Each GNN model is implemented using PyTorch Geometric, with 3 layers and a hidden dimension of 32.

Training Protocol. 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 evaluated two variants of our proposed Why-Inf framework for constraint-based witness generation, along with representative baselines from prominent GNN explanation literature. All methods share the same verification and constraint evaluation pipelines to ensure fair comparison.

ApxIChase. The baseline variant implementing the full two-stage Chase & Backchase pipeline with edge-insertion style candidate generation and top-K witness selection (Algorithm 11). It serves as the reference implementation for correctness and completeness,

ensuring correctness and completeness in the constrained witness discovery process.

HeuIChase. A fast arborescence-based heuristic that replaces the edge-level generation in ApxIChase with Edmonds' arborescence sampling guided by node embedding similarity (Algorithm ??). It significantly accelerates candidate generation while maintaining the same witness verification and window update procedure.

Baselines.We further compare with representative explanation-based methods and a simple constraint-repair baseline:

GNNExplainer [40]. A widely adopted post-hoc explanation method that identifies subgraphs most influential to the model's prediction by optimizing a continuous edge mask through gradient-based learning. In our setting, its selected subgraphs are treated as candidate witnesses for comparison.

PGExplainer [26]. A parameterized probabilistic explainer that learns a neural edge generator to produce explanations conditioned on node embeddings. It serves as a learned baseline for witness discovery, capturing probabilistic dependencies in the latent representation space.

naiveChase. A straightforward yet computationally intensive strategy: it first performs complete constraint repair on the neighborhood of the target node to ensure all applicable rules are satisfied, and then extracts minimal subgraphs consistent with the repaired structure. This baseline enforces all constraints exhaustively to maximize rule coverage, but offers little selectivity and incurs substantial computational overhead, providing an upper bound on constraint coverage but not efficiency.

Evaluation Metrics. We evaluate each method from two perspectives: efficiency and effectiveness.

Efficiency. We measure the average wall-clock time (in seconds) required to complete each experiment. For streaming-style algorithms (e.g., ApxIChase, HeuIChase), this metric covers the full pipeline of candidate generation, verification, and window maintenance across all target nodes or graphs, while for baseline methods, it represents the total runtime needed to generate their explanations under the same experimental configuration.

Effectiveness. We evaluate the quality and faithfulness of generated witnesses using three complementary metrics.
(1) *Fidelity*. We assess how well each witness G_s preserves the model's prediction on the original graph: $\text{fidelity}^-(G_s) = \Pr(M(v_t, G)) - \Pr(M(v_t, G_s))$, where smaller values indicate higher faithfulness. For methods producing candidates(S_k), we report the average fidelity $\frac{1}{K}\sum_{G_s \in S_K} \text{fidelity}^-(G_s)$. To simplify comparison across metrics, we transform this minimization objective into its maximization counterpart,*i.e.,* $1 - fidelity^-$.
(2) *Conciseness*. We quantify the structural compactness of witnesses as $\text{conc}(G_s) = 1 - \frac{|E(G_s)|}{|E(H)|}$, where higher values denote smaller and more focused subgraphs. For multi-candidate settings, we report the mean conciseness over S_k.
(3) *Coverage*. To assess the constraint coverage, we define coverage $= \frac{|\cup_{G_s \in S_K} \Gamma(G_s,k)|}{|\Sigma|}$, no more $\mathcal{S}_{\|}$ which measures the fraction of constraints satisfied collectively by the selected witnesses under budget k. This corresponds to the term in $F(\cdot)$ defined in sec ??. For single-witness baselines, groundedness reduces to the proportion of constraints covered by their single output.

7.2 Experimental Results

Exp-1: Efficiency Comparison. We compare the runtime of all methods, including our proposed ApxIChase, HeuIChase, and MultiIChase, as well as baseline approaches, across all datasets and GNN models. For clarity, method names are abbreviated in all figures as follows: ApxC (ApxIChase), HeuC (HeuIChase), GEX (GNNExplainer), PGX (PGExplainer), and Exh (exhaustive-repair baseline). Unless otherwise specified, we adopt (K=6, k=8, L=2), incompleteness=5%, $|\Sigma|$=20, $\frac{V_t}{V}$=1% as default. Here, *incompleteness* denotes the proportion of missing edges in the input graph relative to its complete version, $|V_t|$ denotes the number of sampled target nodes. For clarity, method names are abbreviated in all figures as follows: when studying the impact of a single factor, we vary that factor while fixing the others at their default values.

Overall Efficiency. Our methods consistently outperform baseline approaches in runtime across all datasets and GNN architectures. As shown in Fig. 8(a), both ApxIChase and HeuIChase achieve substantial speedups compared to the exhaustive constraint enforcement. ApxIChase reduces total runtime by up to 10× on Cora and by over 35× on MUTAG, HeuIChase further improves efficiency through Edmonds-based arborescence generation, achieving up to 500× improvement over exhaustive repair while maintaining comparable fidelity. Among baselines, GNNExplainer remains the fastest method overall, as it performs local mutual information optimization without additional training. PGExplainer, in contrast, requires training on the entire graph, leading to higher runtime on large datasets such as Cora, and failing to scale to BAShape. Overall, ApxIChase and HeuIChase strike a balance between computational efficiency and constraint coverage, demonstrating strong scalability across datasets of different sizes.

Varying $|\Sigma|$. 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 analyze the scalability of different methods with respect to rule set size. As shown in Fig. 8(b), both ApxIChase and HeuIChase exhibit moderate runtime growth when $|\Sigma| \leq 20$, as more constraints increase the number of potential matches that must be checked during the backchase process. When $|\Sigma|$ becomes larger, however, the runtime stabilizes: most constraints no longer match any candidate consequents, resulting in fewer enforceable rules and limited additional cost. HeuIChase shows only a marginal increase throughout, since its Edmonds-based arborescence generation is largely independent of the number of constraints and only lightly influenced by the verification step. By contrast, GNNExplainer and PGExplainer remain flat across all settings because they do not incorporate constraint checking, while the exhaustive-repair baseline shows a steady but high runtime due to its repeated enforcement of all constraints regardless of applicability.

Varying Number of Target Nodes. 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 ApxIChase and

HeuIChase exhibit an approximately linear increase in total runtime as the number of target nodes grows. This is expected, since each additional target node triggers a separate round of candidate generation, verification, and constraint grounding. Despite the growth, HeuIChase remains consistently lightweight—under 15 seconds even at 5% sampling—demonstrating its strong scalability and efficiency in multi-target settings. ApxIChase also scales gracefully, maintaining stable per-target cost as the explanation workload increases. In contrast, the exhaustive baseline grows rapidly with $|V_t|$, reflecting the high cost of repeatedly enforcing all constraints across multiple neighborhoods. GNNExplainer shows a similar but milder upward trend, while PGExplainer remains nearly constant due to its global training paradigm. Overall, both ApxIChase and HeuIChase exhibit excellent scalability with respect to the target node ratio, achieving orders of magnitude lower runtime than exhaustive repair while maintaining grounded witness generation.

Varying GNN *layer L*. We investigate how model depth (*i.e.*,, the number of GNN layers or equivalently the hop count L) affects runtime efficiency on the Cora dataset. We keep the architecture family fixed as GCN and vary the number of layers (equivalently, the neighborhood hop count) $L \in \{1, 2, 3\}$, with all other factors held constant. As shown in Fig. 8(d), both ApxIChase and HeuIChase exhibit sublinear growth in runtime with respect to model depth, demonstrating their scalability with increasing neighbor sizes. In contrast, the exhaustive-repair baseline shows a sharp increase in cost (from 142s to 3,566s) as L grows, indicating the exponential overhead of full 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), ApxIChase exhibits a mild downward trend as incompleteness increases. This behavior can be attributed to its candidate generation mechanism: a higher degree of missingness effectively reduces the search space of valid edge insertions, allowing faster convergence during witness construction. In contrast, the exhaustive-repair baseline shows the opposite pattern—its runtime grows steadily with higher incompleteness, since a less complete graph introduces more constraint violations that must be repaired, significantly increasing the overall enforcement overhead. HeuIChase remains largely unaffected, as its arborescence-based generation explores a fixed number of candidate edges regardless of missingness. GNNExplainer and PGExplainer also show little variation, 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 ApxIChase and HeuIChase exhibit stable performance across different architectures, with runtime difference primarily reflecting the computational complexity of the underlying GNN forward passes. HeuIChase 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 Comparison. We next evaluate the overall quality of witness generation in terms of constraint satisfaction and fidelity to model predictions. All experiments are conducted under the default configuration of (K=6, k=8, L=2, incompleteness=5%, $|\Sigma|$=20, $|V_t|/|V|$=1%), unless otherwise specified. When studying the impact of a single factor, we vary that factor while keeping all others fixed at their default values. Unless otherwise noted, these factor-specific experiments are conducted on the Cora dataset. We first analyze constraint satisfaction by examining overall coverage and its variation under different factors, including budget k, $|\Sigma|$, model depth L, and incompleteness. We then assess model fidelity to evaluate how well the grounded witnesses preserve the predictive behavior of the original GNNs.

Overall Coverage. We first compare the overall constraint coverage across all datasets under the default configuration, as shown in Fig. 9(a). Both ApxIChase and HeuIChase consistently achieve high coverage, with ApxIChase reaching near-complete grounding on ATLAS, Cora, and BAShape, and HeuIChase performing comparably except on Cora where its tree-like witnesses limit the ability to satisfy cyclic constraints. GNNExplainer and PGExplainer, which do not incorporate explicit constraint evaluation, 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 incidentally satisfy some of the acyclic constraint patterns. In contrast, the exhaustive-repair baseline attains full coverage on all datasets, as it enforces all rules before witness generation, serving as an upper bound for achievable coverage.

Coverage Varying 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 ApxIChase 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, HeuIChase 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 Varying $|\Sigma|$. We vary the size of constraint set $|\Sigma| \in \{10, 20, 30, 40, 50\}$ while fixing other factors at their default values. As shown in Fig. 9(c), both ApxIChase and HeuIChase exhibit

a gradual decline in coverage as $|\Sigma|$ increases. This is expected, since under a fixed budget each method can only ground a limited number of constraints, causing the overall ratio to decrease as the denominator grows. The decline, however, is sublinear — many additional constraints are either redundant or partially overlapping with previously satisfied ones, allowing the algorithms to maintain moderate coverage even for large $|\Sigma|$. In comparison, GNNExplainer and PGExplainer remain unaffected, as they do not incorporate constraint grounding, while the exhaustive baseline maintains full coverage by pre-enforcing all rules.

Coverage Varying 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 ApxIChase and HeuIChase exhibit increasing coverage with larger L, since 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. By contrast, GNNExplainer and PGExplainer remain flat at zero coverage, as they lack constraint evaluation, while the exhaustive-repair baseline maintains full coverage across all depths due to complete rule enforcement prior to witness generation.

Coverage Varying Incompleteness. We investigate how varying levels of graph incompleteness (*i.e.*,, 5%–20% missing links) affect constraint coverage while keeping other factors fixed. As shown in Fig. 9(e), both ApxIChase and HeuIChase exhibit a gradual decline in coverage as incompleteness increases. As anticipated, since under a fixed budget, a higher fraction of missing edges reduces the available structural context, making it more difficult to ground all applicable constraints. The decrease is more pronounced for HeuIChase due to its tree-like witness structure, as the removal of edges limits the variety of its arborescence-shaped candidates, reducing their ability to collectively cover diverse constraint patterns. The other methods remain largely unaffected by incompleteness.

overall Fidelity. We report the overall 1−fidelity$^-$ scores (the higher, the better) across all datasets under the default configuration, as shown in Table 9(f). Our ApxIChase and HeuIChase consistently achieve the highest fidelity values on all datasets, demonstrating that constraint-grounded witnesses better preserve model predictions compared to baselines. Notably, the improvement is most pronounced on datasets with well-defined structural rules, such as MUTAG and BAShape. Once the relevant constraints are grounded (e.g., aromatic rings or house motifs), the candidate witnesses naturally align with the true functional or structural patterns, leading to near-perfect agreement with the model's original decision (1−fidelity$^-$ $\approx$ 0.99). Interestingly, both ApxIChase and HeuIChase occasionally outperform the exhaustive-repair baseline. This suggests that fully enforcing all constraints may not always improve model faithfulness—instead, it can introduce artificial edges or structures that were absent in the original graph, thereby degrading the model's predictive behavior. In contrast, GNNExplainer and PGExplainer lack explicit constraint enforcement and thus may focus on locally influential but semantically incomplete subgraphs, resulting in lower fidelity scores. These results highlight the strong 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 (paraIChase) under both large-scale real and synthetic graphs, with full algorithmic details provided in the full version. cite full version 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.

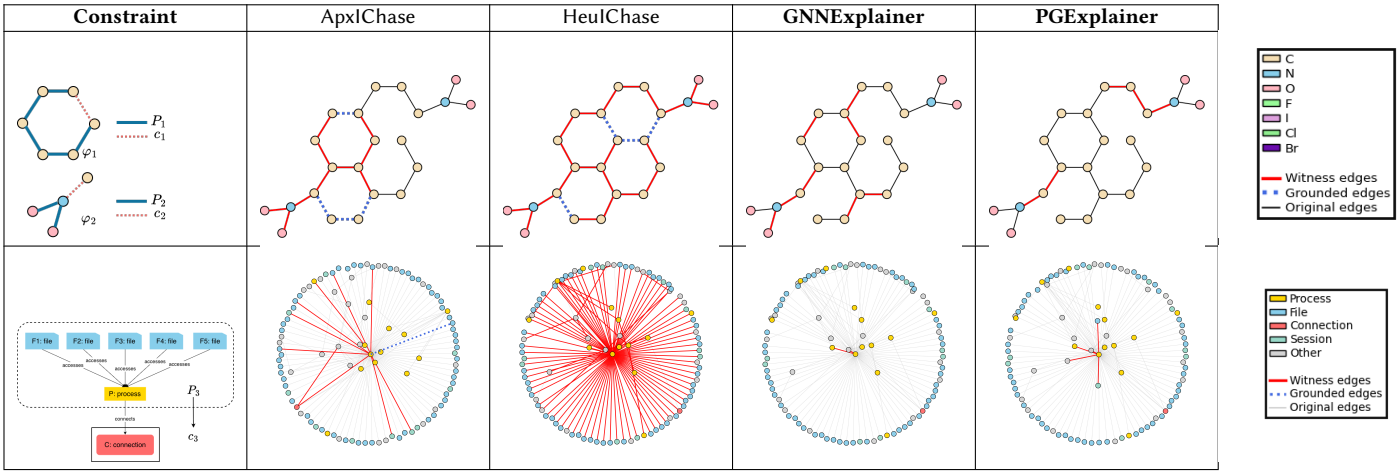
OGBN-Papers100M. Figure 10(a) shows that both ApxIChase and HeuIChase achieve clear speedups when scaling from 4 to 20 processors. Despite highly unbalanced subgraph sizes (some exceeding 80K edges), runtimes remain tolerable—dropping from 13.9K⟶2.6Ks for ApxIChase and 282 ⟶ 54s for HeuIChase. PGExplainer and exhaustive-repair baseline fail to scale, while GNNExplainer stays the fastest but saturates beyond p=10.

Tree-Cycles. We further evaluate scalability on the synthetic Tree-Cycles dataset under three controlled configurations. (1)*Varying graph size:* As the overall graph grows from 1.1M to 1.4B edges ($\approx$ 25 GB in memory), both ApxIChase and HeuIChase show moderate runtime growth and remain tractable, while the exhaustive-repair baseline exceeds 28, 800 s (> 8 h) with 20 processors and incurs substantial memory inflation. In our parallel setting, each processor handles 4˜6 L-hop subgraphs ($\approx$ 4.6K edges per task). Under exhaustive repair, enforcing all candidate constraints within each subgraph can temporarily double or triple its size, leading to up to $\approx$ 1 GB memory per processor ($\approx$ 20 GB total). In contrast, ApxIChase and HeuIChase only operate on compact witnesses—typically 20˜40% of the subgraph size—without expanding the original structure, keeping the per-processor footprint below 200˜300 MB. Overall, our methods reduce memory overhead by roughly 4˜5× compared with exhaustive repair, while maintaining stable runtime and balanced workload distribution. (2)*Varying processor count:* Fixing the 1.4B-edge graph and 100 target nodes, we vary the number of processors from 4 to 20 (Fig.10(c)). Both ApxIChase and HeuIChase achieve sublinear yet consistent speedups, primarily limited by coordination and I/O overheads during subgraph extraction and result aggregation. Nevertheless, the makespan decreases by over 5×, demonstrating effective load balancing and low communication contention even under high parallelism. (3)*Varying query load:* On the same 1.4B-edge graph with 20 processors, we increase the number of target nodes from 100 to 500 (Fig.10(d)). Runtime grows nearly linearly with query load since subgraph extraction and constraint grounding dominate the total cost, but the growth rate remains predictable and well aligned across all algorithms. Overall, our methods achieve 1˜2 orders of magnitude runtime improvement and substantially lower memory overhead compared with exhaustive repair, while maintaining stable scalability across graph size, processor count, and query complexity.

Exp-4: Case Studies.

Case Study I: Witness Generation in Incomplete Molecular Graphs. Table. 2(Row 1) illustrates a representative example from the MUTAG dataset, demonstrating how constraint grounding enhances

Table 2: Case study



the interpretability of witnesses under graph incompleteness. Two constraints are considered on the left: (1) $P_1 \rightarrow c_1$ represents a ring-closure rule, enforcing 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 are missing, our ApxIChase identifies a witness subgraph (red edges) that already grounds the second constraint (oxygen–nitrogen linkage), and under a budget of k=4, it adds 4 grounded edges (blue dotted) to enforce two benzene rings, recovering the missing aromatic structures in a chemically valid manner. HeuIChase similarly constructs a spanning-tree–like witness that grounds the same functional linkage rule and enforces ring-closure constraint within the available budget. In comparison, both GNNExplainer and PGExplainer produce fragmented subgraphs that lack chemical interpretability and fail to ground any of the predefined constraints, underscoring the necessity of constraint-aware witness generation.

Case Study 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 attack structures 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)ApxIChase 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→F1–F5) and exfiltration (backdoor→C). The result provides investigators with a complete, constraint-aligned explanation of theft and transmission. (2)HeuIChase also recovers all six critical nodes but embeds them in a dense arborescence that mixes unrelated files and processes due to similarity-based edge weighting. Although φ_3 holds, the structure requires manual pruning to isolate the true attack path. (3)GNNExplainer detects edges to only F1 and F4, omitting F2–F3–F5 and C; the missing consequent and partial antecedent violate φ_3, leaving an incomplete narrative that suggests benign file access. (4)PGExplainer approximates coverage but deviates from the constraint by connecting to auxiliary sessions and excluding some files (F2, F3, F5). Its probabilistic sampling preserves model fidelity yet breaks the discrete antecedent–consequent structure, yielding a diffuse, weakly grounded witness.

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. A future topic is to extend our algorithms over distributed and dynamic networks, and temporal graph models.

REFERENCES

[1] Abdulellah 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.

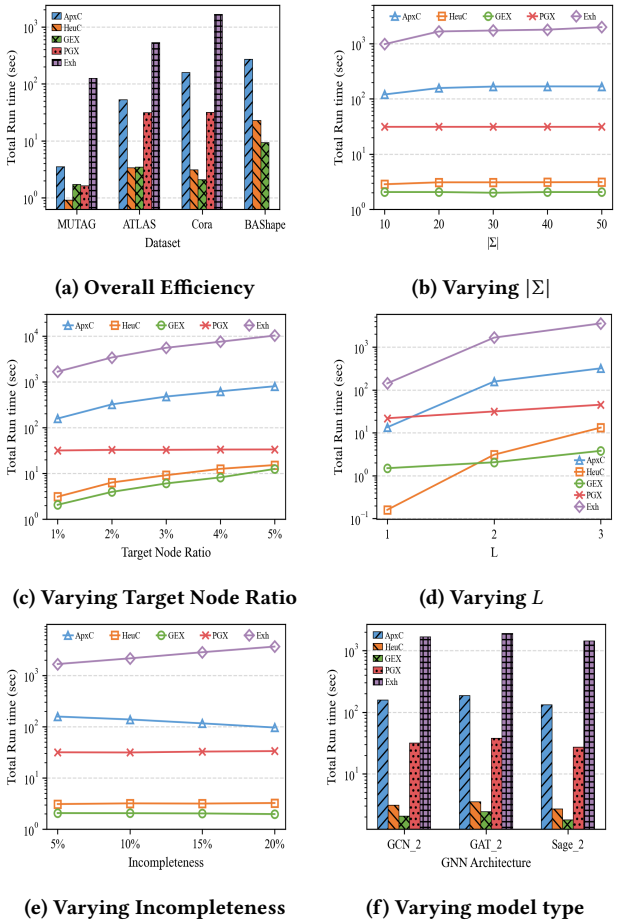
[2] Burouj Armgaan, Manthan Dalmia, Sourav Medya, and Sayan Ranu. 2024. GraphTrail: Translating GNN predictions into human-interpretable logical rules. *Advances in Neural Information Processing Systems* 37 (2024), 123443–123470.

[3] Michael Benedikt, George Konstantinidis, Giansalvatore Mecca, Boris Motik, Paolo Papotti, Donatello Santoro, and Efthymia Tsamoura. 2017. Benchmarking the chase. In *PODS*. 37–52.

[4] Peter Berger, Gabor Hannak, and Gerald Matz. 2020. Efficient graph learning from noisy and incomplete data. *IEEE Transactions on Signal and Information Processing over Networks* 6 (2020), 105–119.

[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.

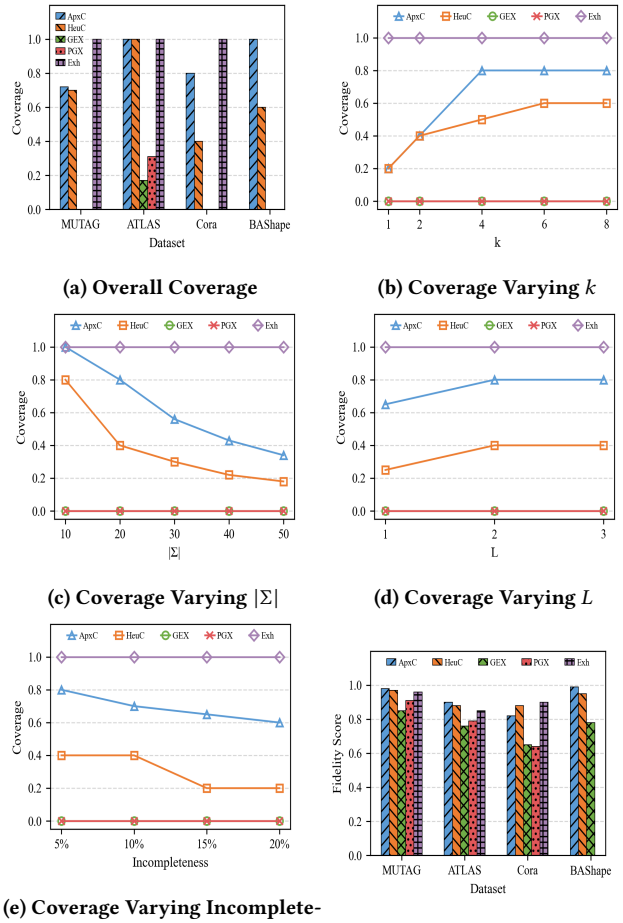


(a) Overall Efficiency (b) Varying $|\Sigma|$

(c) Varying Target Node Ratio (d) Varying L

(e) Varying Incompleteness (f) Varying model type

Figure 8: Efficiency of Why-Inf



(a) Overall Coverage (b) Coverage Varying k

(c) Coverage Varying $|\Sigma|$ (d) Coverage Varying L

(e) Coverage Varying Incompleteness (f) Overall Fidelity

Figure 9: Effectiveness of Why-Inf

[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] Boris Glavic and Gustavo Alonso. 2009. Perm: Processing provenance and data on the same data model through query rewriting. In *2009 IEEE 25th International Conference on Data Engineering*. IEEE, 174–185.

[16] Todd J Green, Grigoris Karvounarakis, and Val Tannen. 2007. Provenance semirings. In *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 31–40.

[17] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *NeurIPS* (2017).

[18] 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).

[19] Sheng Huang et al. 2015. Why is this ranked high? Explainability of search result rankings. In *SIGMOD*.

[20] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[21] Ryoji Kubo and Djellel Difallah. 2025. RAW-Explainer: Post-hoc Explanations of Graph Neural Networks on Knowledge Graphs. *arXiv preprint arXiv:2506.12558* (2025).

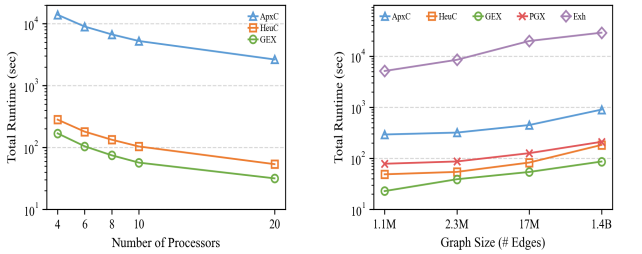
[22] Wangchao Le, Anastasios Kementsietsidis, Songyun Duan, and Feifei Li. 2012. Scalable multi-query optimization for SPARQL. In *2012 IEEE 28th International Conference on Data Engineering*. IEEE, 666–677.

[23] Wanyu Lin, Hao Lan, and Baochun Li. 2021. Generative causal explanations for graph neural networks. In *International conference on machine learning*. PMLR, 6666–6679.

[24] 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.

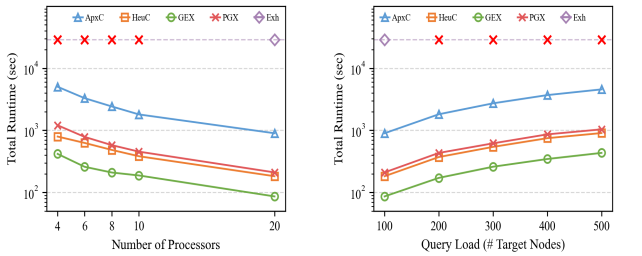
[25] Ana Lucic, Maartje A Ter Hoeve, Gabriele Tolomei, Maarten De Rijke, and Fabrizio Silvestri. 2022. Cf-gnnexplainer: Counterfactual explanations for graph neural networks. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 4499–4511.

[26] 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.



(a) Runtime Varying Number of Processors on OGBN-Papers100M

(b) Runtime Varying Graph Size on Tree-Cycles



(c) Runtime Varying Number of Processors on Tree-Cycles

(d) Runtime Varying Query Load on Tree-Cycles

Figure 10: Scalability Test

[27] 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.

[28] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. 2000. Automating the construction of internet portals with machine learning. *Information Retrieval* (2000).

[29] Alexandra Meliou, Wolfgang Gatterbauer, Joseph Y Halpern, Christoph Koch, Katherine F Moore, and Dan Suciu. 2010. Causality in databases. *IEEE Data Eng. Bull.* 33, 3 (2010), 59–67.

[30] Haoran Miao, Eugene Wu, Samuel Madden, and Semih Salihoglu. 2012. Towards unified provenance detection for why-not and why-so queries. In *SIGMOD*.

[31] Paolo Pareti, George Konstantinidis, Timothy J Norman, and Murat Şensoy. 2019. SHACL constraints with inference rules. In *International Semantic Web Conference*. Springer, 539–557.

[32] Xuguang Ren and Junhu Wang. 2016. Multi-query optimization for subgraph isomorphism search. *Proceedings of the VLDB Endowment* 10, 3 (2016), 121–132.

[33] Marcus Schaefer and Christopher Umans. 2002. Completeness in the polynomial-time hierarchy: A compendium. *SIGACT news* 33, 3 (2002), 32–49.

[34] Larissa C Shimomura, George Fletcher, and Nikolay Yakovets. 2020. GGDs: graph generating dependencies. In *CIKM*.

[35] Qi Song, Mohammad Hossein 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.

[36] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, Yoshua Bengio, et al. 2017. Graph attention networks. *stat* (2017).

[37] Minh Vu and My T Thai. 2020. Pgm-explainer: Probabilistic graphical model explanations for graph neural networks. *Advances in neural information processing systems* 33 (2020), 12225–12235.

[38] 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.

[39] Eugene Wu and Samuel Madden. 2013. Provenance for ranking and recommendations. In *CIDR*.

[40] 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).

[41] 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.

[42] Wenyue Zhao, Yang Cao, Peter Buneman, Jia Li, and Nikos Ntarmos. 2024. Automating Vectorized Distributed Graph Computation. *Proceedings of the ACM on Management of Data* 2, 6 (2024), 1–27.

[43] 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).

[44] 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.

[45] Chuanyu Zong and Chengwei Zhang. 2024. Effectively answering why questions on structural graph clustering. *Applied Soft Computing* 154 (2024), 111405.

9 APPENDIX

9.1 Appendix A: Proofs

Hardeness Analysis.

Verification of Why-Inf. To understand the hardness of witness computation under constraints, we start by investigating a decision problem called verification. Given a subgraph G_s of G, a GNN model M, and a set of data constraints Σ, the verification problem is to decide if (1) G_s is either a factual or a counterfactual witness of M over G, and (2) if $G_s \models \Sigma$.

Theorem 4: *The verification problem for* Why-Inf *is* coNP*-hard.* □

Proof sketch: The hardness carries over from the hardness of the validation problem of graph functional dependencies, which is a special case that G_s has passed the PTIME test of witness by calling a GNN inference function, and to decide whether $G_s \models \varphi$ as a single graph functional dependency. □

Hardness of Provenance generation.

Theorem 5: *The* Why-Inf *problem is already* Π_2^P *even for fixed input* G, M *and* V_t. □

Proof sketch: A reduction from min-max Clique problem [33], 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 Why-Inf that aims to find if there exists a witness from a finite set of graphs (which are all validated 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 Σ of at most k constraints, with patterns P as a size i clique, and $i \in [1, k]$ for the i-th constraint. Moreover, as G and M are fixed (with L number of layers), there are polynomial number of subgraphs from G to be verified. If there exists a witness that satisfies Σ, it indicates a graph exists from a set of input graphs that contains a largest clique with size at least k; and vice versa. As min-max Clique is known to be Π_2^P-complete, the hardness of Why-Inf carries over. □

Proof of Theorem ??.

Detailed proof of Axiomatic Analysis. We show that the function $F(\mathcal{G}_s)$ is scale invarance, consistent, non-monotonic, and independent.

Proof of Theorem 4. The hardness carries over from the hardness of the validation problem of graph functional dependencies, which is a special case that G_s has passed the PTIME test of witness by

Algorithm 4 : **Procedure** UpdateWK (W_k, G_s, Σ, B)

$\Gamma_{G_s} \leftarrow \Gamma(G_s, B)$
$\Gamma_{W_k} \leftarrow \bigcup_{(G'_s,\cdot)\in W_k} \Gamma(G'_s, B)$
if $\Gamma_{G_s} = \emptyset$ **then**
  return
if $|\Gamma_{G_s} \setminus \Gamma_{W_k}| = 0$ **then**
  return
$\Delta F(G_s) \leftarrow \alpha \cdot \mathsf{conc}(G_s) + \beta \cdot \mathsf{rpr}(G_s) + \gamma \cdot \frac{|\Gamma_{G_s} \setminus \Gamma_{W_k}|}{|\Sigma|}$
if $|W_k| < k$ **then**
  heappush(W_k, $(G_s, \Delta F(G_s))$) **else**
  let $(G_{\min}, \Delta F_{\min})$ be the heap-min of W_k
  if $\Delta F(G_s) > \Delta F_{\min}$ **then**
    heappushpop(W_k, $(G_s, \Delta F(G_s))$); update coverage stats

Figure 11: Procedure UpdateWK

calling a GNN inference function, and to decide whether $G_s \models \varphi$ as a single graph functional dependency.

Proof of Theorem 5. A reduction from min-max Clique problem [33], 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 Why-Inf that aims to find if there exists a witness from a finite set of graphs (which are all validated 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 Σ of at most k constraints, with patterns P as a size i clique, and $i \in [1, k]$ for the i-th constraint. Moreover, as G and M are fixed (with L number of layers), there are polynomial number of subgraphs from G to be verified. If there exists a witness that satisfies Σ, it indicates a graph exists from a set of input graphs that contains a largest clique with size at least k; and vice versa. As min-max Clique is known to be Π_2^P-complete, the hardness of Why-Inf carries over.

9.2 Appendix B: Algorithms and Analysis

Details of I-Chase. The I-Chase procedure extends Chasealgorithm to enforce each graph constraint $\varphi \in \Sigma$ in $G^L(v_t)$ and derive subgraphs $G_\varphi = G_s \oplus e_c$, where G_s is a subgraph of G hat matches P, and e_c is a single edge e_c that matches triple pattern c, where e_c is either inserted (enforced), or originally exists in G. For each such subgraph $G_\varphi = G_s \oplus e_c$, it invokes a verification process that performs an inference of M over G_φ to checks whether G_φ serve as a witness for the inference query $Q(M, v_t, G)$. If so, it packs G_φ as a 0-grounded witness by φ in $G^L(v_t)$.

For each $\varphi \in \Sigma$, it enumerates all the 0-grounded witnesses. It then computes the quality of each 0-grounded witnesses, and returns top-K ones with maximized

Optimization: Pruning of Σ. We reformulate constraint evaluation as a multi-instance, multi-pattern workload, where many candidate subgraphs and many TGDs must be checked concurrently. Inspired by vectorized multi-instance graph computation and multi-query optimization [22, 32, 42], we share computation across rules and candidates rather than processing each check in isolation. Concretely, for TGDs of the form $\tau : \varphi \to \psi$. We group rules by their consequents ψ: for a consequent ψ, we detect matches of ψ once and then drive the backchase for all rules in the same bucket. A SMS maintains inverted indexes and cached partial matches for common atoms; when the layered edge-insertion strategy introduces a new edge, only the postings involving its incident nodes are updated incrementally, avoiding full re-evaluation. Neighborhood and attribute indexes further allow constant-time filtering by relation type or node attributes.

We then exploit the following pruning lemmas to avoid redundant checks:

Lemma 6: *Fix a consequent ψ and two bodies with containment $\varphi_i \supseteq \varphi_j$. If* $\mathrm{rep}(G_s, \psi \to \varphi_i) \le B$ *then* $\mathrm{rep}(G_s, \psi \to \varphi_j) \le B$. *Hence tests for $\psi \to \varphi_j$ can be skipped once $\psi \to \varphi_i$ is grounded.* □

Proof sketch: Any realization of the larger antecedent φ_i subsumes a realization of φ_j; removing atoms cannot increase the number of missing edges. Thus the minimal backchase cost to φ_j is $\le$ that to φ_i. If the latter fits in B, so does the former. □

Lemma 7: *If rep$(G_s, \varphi \to \psi) > B$, then for every super-antecedent $\varphi' \supseteq \varphi$, rep$(G_s, \varphi' \to \psi) > B$ also holds. Thus all such rules are prunable.* □

Proof sketch: Adding more atoms to the antecedent can only increase (never decrease) the number of missing edges required for repair. Hence if φ already exceeds the repair budget, so do all its super-bodies. □

Lemma 8: *Let Σ' be the set of grounded rules already proved over G_s, each with a repair cost not exceeding B. Let* $\mathrm{cl}(\Sigma', B)$ *be the closure of Σ' under budget-aware derivation principles (e.g., transitivity, conjunctive combination), where the composed repair cost is the sum or union of the respective repairs. If $\tau \in \mathrm{cl}(\Sigma', B)$, then $G_s \models \tau$ can be concluded without explicit subgraph matching.* □

Proof sketch: Consider two representative cases:

Transitivity. Suppose $\psi \to \varphi_1 \in \Sigma'$ with $\mathrm{rep}(G_s, \psi \to \varphi_1) = c_1$, and $\varphi_1 \to \varphi_2 \in \Sigma'$ with $\mathrm{rep}(G_s, \varphi_1 \to \varphi_2) = c_2$. Then the combined grounding justifies $\psi \to \varphi_2$ with cost at most $c_1 + c_2$. If $c_1 + c_2 \le B$, we may conclude that $G_s \models (\psi \to \varphi_2)$ without additional subgraph evaluation.

Conjunctive combination. Suppose $\psi \to \varphi_1 \in \Sigma'$ with $\mathrm{rep}(G_s, \psi \to \varphi_1) = c_1$, and $\psi \to \varphi_2 \in \Sigma'$ with $\mathrm{rep}(G_s, \psi \to \varphi_2) = c_2$. Then the stronger rule $\psi \to (\varphi_1 \wedge \varphi_2)$ holds with combined cost at most $c_1 + c_2$. If this total does not exceed B, we can skip explicit verification.

Thus, whenever $\tau \in \mathrm{cl}(\Sigma', B)$, it is already justified by budget-safe compositions of existing grounded rules, and need not be explicitly matched again. □

Together, these lemmas guarantee that constraint evaluation proceeds incrementally with maximal reuse: consequent matches are shared across rules, postings are updated locally, and redundant verifications are pruned. As a result, the backchase phase avoids repeated full scans over candidates and constraints, substantially reducing overhead while preserving correctness of ApxIChase.

Correctness & Cost Analysis of ApxIChase. We prove that ApxIChase always terminates and returns feasible solutions that maximize constraint satisfaction. (1) The edge-insertion process is monotone: each step adds one edge from the finite L-hop set E_H, and no edge is ever removed. Hence the procedure terminates within at most $|E_H|$ iterations. (2) For each candidate subgraph G', the procedure VerifyWitness ensures that the prediction of v_t is preserved before admitting G' into the top-k window. Therefore every subgraph retained in $\mathcal{S}_k$ is feasible. (3) The procedure UpdateWindow dynamically maintains $\mathcal{S}_k$ by greedy replacement, ensuring that the final output consists of k feasible subgraphs that jointly maximize constraint satisfaction, together with the active constraints Σ^* they cover.

In the chase phase, each edge is removed at most once, generating $O(|E_H|)$ candidate subgraphs; each requires a call to VerifyWitness, incurring total cost $O(|E_H| \cdot T_m)$, where T_m is the inference cost of the underlying model (polynomial in the size of the input subgraph). Constraint evaluation over $|\mathcal{S}|$ candidates and $|\Sigma|$ constraints has worst-case complexity $O(|\mathcal{S}| \cdot |\Sigma|)$, but shared match stores and neighborhood/attribute indexes enable incremental evaluation and reuse of partial results, reducing practical cost close to $O(|\mathcal{S}| + |\Sigma|)$. Maintaining the top-$k$ window adds $O(|\mathcal{S}| \log k)$ time for heap updates. Overall, the worst-case time complexity is $O(|E_H| \cdot T_m + |\mathcal{S}| \cdot |\Sigma| + |\mathcal{S}| \log k)$, with practical performance substantially improved by shared evaluation and pruning techniques.

Theorem 9: *Let*

$$F_{\text{tot}}(S) = \alpha \sum_{G_s \in S} \text{conc}(G_s) + \beta \sum_{G_s \in S} \text{rpr}(G_s) + \gamma \frac{|\Gamma(S)|}{|\Sigma|},$$

where $\Gamma(S) = \bigcup_{G_s \in S} \Gamma(G_s)$ *and* $\alpha, \beta, \gamma \geq 0$. *Then* F_{tot} *is a monotone submodular set function under the cardinality constraint* $|S| \leq k$. □

Proof sketch: The first two terms are modular (i.e., additive across elements), hence both monotone and submodular. For the coverage term, define the marginal gain of adding a candidate G_s to a set S as $\Delta(G_s \mid S) = |\Gamma(S \cup \{G_s\})| - |\Gamma(S)|$. It is immediate that for $A \subseteq B$, $\Delta(G_s \mid A) \geq \Delta(G_s \mid B)$, i.e., adding G_s yields diminishing returns as the set grows. Thus $|\Gamma(S)|$ is monotone submodular. A nonnegative linear combination of a modular function and a submodular function remains submodular. Since each component is also monotone, the overall F_{tot} is monotone submodular. □

Lemma 10: *Let* $\text{OPT} = \max_{|S| \leq k} F_{\text{tot}}(S)$ *denote the optimal value under the cardinality constraint. Then the greedy selection rule that maximizes the marginal gain* ΔF_{tot} *at each step yields a set* S_k *such that* $F_{\text{tot}}(S_k) \geq (1 - 1/e) \cdot \text{OPT}$. □

Proof sketch: Our implementation admits a candidate G_s only if it introduces new coverage, *i.e.,* $\Gamma(G_s) \setminus \Gamma(S) \neq \emptyset$. Under this gating, the greedy selection coincides with the standard greedy algorithm for the coverage objective $|\Gamma(S)|$, which preserves the $(1 - 1/e)$ guarantee for coverage. The modular quality terms (α, β) serve as tie-breakers among candidates with the same new coverage and can only improve the final F_{tot} value. □

Algorithm 5 : paraIChase: Parallel Witness Generation

Input: Graph G, constraint set Σ, model M, target node set V_T, hop L, window k, budget B, #processors p, algorithm ALG $\in$ {ApxIChase, HeuIChase, Exh, GEX, PGX}

Output: Per-target results $\{(\Sigma^*(v), \mathcal{S}_k(v)) \mid v \in V_T\}$ and aggregated statistics

$\mathcal{H} \leftarrow \emptyset$ ▷ coordinator: compute per-target L-hop subgraph
for all $v \in V_T$ **do**
  $H_v \leftarrow G_v^L$; $\mathcal{H} \leftarrow \mathcal{H} \cup \{(v, H_v)\}$
$P \leftarrow \text{Partition}(V_T, \mathcal{H}, p)$ ▷ load balance by L-hop size
for $i = 1$ **to** p **do in parallel**
  $\mathcal{R}_i \leftarrow \emptyset$
  for all $v \in P_i$ **do**
    $(\Sigma^*(v), \mathcal{S}_k(v)) \leftarrow \text{Run\_Algorithm}(v, \text{ALG})$
    $\mathcal{R}_i \leftarrow \mathcal{R}_i \cup \{(v, \Sigma^*(v), \mathcal{S}_k(v))\}$
$\mathcal{R} \leftarrow \bigcup_{i=1}^{p} \mathcal{R}_i$
return $\mathcal{R}$

Figure 12: Algorithm paraIChase**: Parallelized witness generation framework with load-balanced subgraph distribution.**

9.3 Appendix C: Complementary Experimental Study and Discussion

Parallel Algorithm. Algorithm 12 outlines the parallel version of our witness generation framework. A coordinator process induces L-hop subgraphs for all target nodes and partitions them across p processors using load-balanced assignment based on subgraph size. Specifically, we estimate each target's cost as $c(v) = |E(H_v)|$ and perform assignment via a *First-Fit Decreasing* bin-packing with a min-heap over current worker loads, which effectively mitigates stragglers caused by heavy neighborhoods. Each processor independently runs one of the scalable base algorithms ALG $\in$ {ApxIChase, HeuIChase, Exh, GEX, PGX} on its assigned subgraphs to generate witnesses and record constraint coverage. Global parameters (M, Σ, k, B, L) are shared read-only across all processors, and workers obtain local subgraphs via indexed slicing to avoid redundant data loading and excessive memory usage. Candidate verification is performed in mini-batches within each worker, which reduces redundant inference calls and improves overall computational throughput. The coordinator finally aggregates per-target results to compute overall efficiency and effectiveness metrics.