Grounding Provenance for Graph Inference with Data Constraints

Haolai Che, Mingjian Lu, Yangxin Fan, Hanchao Ma, Yinghui Wu
Case Western Reserve University
Cleveland, Ohio, USA
{hxc859,mxl1171,yxf451,hxm382,yxw1650}@case.edu

ABSTRACT

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

PVLDB Reference Format:
Haolai Che, Mingjian Lu, Yangxin Fan, Hanchao Ma, Yinghui Wu.
Grounding Provenance for Graph Inference
with Data Constraints. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/anthonyche/Grounded_Witness.

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 $Q(M, V_T, G)$.

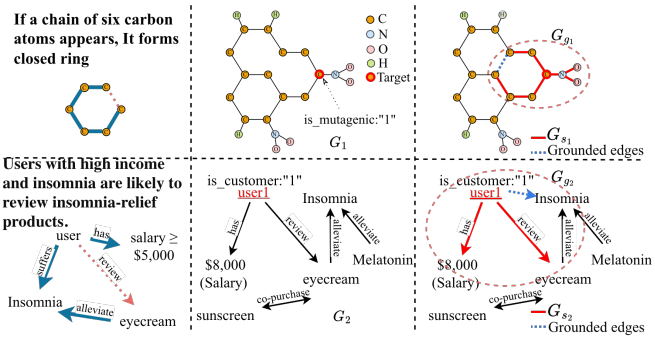


Figure 1: Grounding graph inference analysis with genuine evidence declared in data constraints.

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

Example 1: We illustrate two scenarios as shown in Fig. 1.
(1) **[Molecular Property Analysis]**. Chemists heavily rely on empirical rules (facts) or requirements of occurrences of chemically valid structures for molecular analysis, e.g., carbon-rings. Such a rule can be specified by a data constraint φ_1: *"If a chain of six carbon atoms appears (P_1), it forms a closed ring (c_1)"*. G_1 illustrates a molecule labeled as mutagenic, where the red node is classified by a 3-layer GCN M as a *mutagenic atom-i.e.,* an atom that contributes to the molecule's mutagenicity. There are missing edges in $G1$: two benzene-like structures should have been formed as a stable two 6-atom rings; instead, they are presented as a 10-atom ring. A direct

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

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

(2) **[Web recommendation]**. Consider another scenario that arises in social recommendation systems. A typical customer behavior pattern 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 a specific type of eye cream.

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

The above cases highlight the need to make graph inference grounded with real-world data evidence, which data constraints may suggest. However, real-world graphs may also contain missing edges, making potential data constraints trivially satisfied or not applicable. For example, the data constraint φ_2 cannot be directly enforced to ground "review" relation due to the missing edge "pre-condition" from user to "Insomnia". This calls for an effective method to generate subgraphs that can both clarify the output of inference queries, and also make it grounded in the evidence suggested by data constraints.

Example 2: Following example 1, a more desired provenance structure for the molecular property analysis scenario, as shown in Fig. 1, can be a subgraph G_{g_1} enclosed by the pink dotted oval, including a grounded edge (blue dotted) added to close the 6-atom ring, reconstructing the benzene ring. This makes the output better aligned with the chemically valid evidence. Similarly, a more comprehensive interpretation of the purchase intention of the user can be captured by a graph that includes the red edges from G_{s_2}, and a new edge (blue dotted), revealing the relation between the user and insomnia. This helps the analyst to find a more reasonable recommendation as "Melatonin" (sleep aid), better aligned with the true purchase attention, enabled by the grounded interpretation. On the other hand, as φ_2 cannot be applied to enforce the missing edge, how to derive such a structure? □

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

Constraint-based Provenance for graph inference. Given a graph G with a test node v_t, 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 (a "grounded" provenance graph) that can suggest

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

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

Related work. We summarize related work below.

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

Recent work has extended provenance to graph queries. [36] 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 provenance for graph inference, aiming to explain model predictions by exploiting graph data constraints.

GNN explanation. A growing body of research has focused on explaining the predictions of GNNs, aiming to identify the most influential substructures that contribute to a model's output. Existing post-hoc methods can be broadly categorized into perturbation-based (e.g., GNNExplainer [27, 40]), surrogate-based (e.g., PGMExplainer [38]), and causality-inspired approaches (e.g., GEM [24], RCExplainer [39], OrphicX [25]). Recent work has attempted to improve interpretability through more structured reasoning, yet mostly stop at representing subgraphs into logic rules rather than exploiting the latter [3, 23, 41]. While these methods differ in how they estimate subgraph importance—via masking, generative modeling, or causal inference—they all primarily focus on optimizing for behavioral fidelity, *i.e.,* how well the explanation aligns with

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

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

2 INFERENCE, CONSTRAINTS AND WITNESS

2.1 Graph Inference Queries

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

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

Graph Neural Networks. GNNs [11, 21] 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 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 $M(v, G)$, for the output of a GNN M at a node $v \in V$.

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

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

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

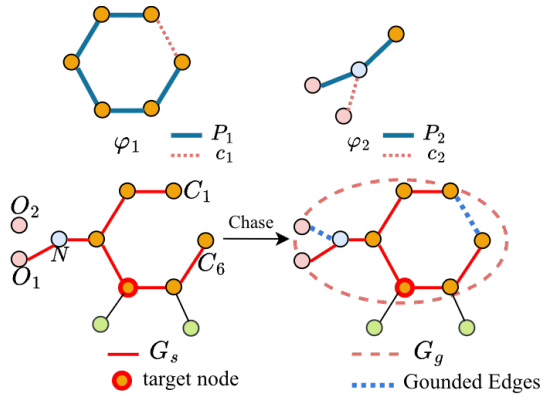


Figure 2: Two data constraints are illustrated: $\varphi_1 = (P_1, c_1)$ enforces a carbon ring, and $\varphi_2 = (P_2, c_2)$ completes a missing bond between nitrogen and oxygen. During Chase, a subgraph G_s matching P_1 and P_2 (red edges) is identified, and the missing bonds (blue dotted) are added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

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.

(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 [14], graph functional [15], or generation dependencies [35], or SHACL [32].

Enforcement. Chase [4, 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 [28], 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: Consider G_s Fig. 2, and let data constraint $\varphi_1 = (P_1, c_1)$ and $\varphi_2 = (P_2, c_2)$, where $P_1 = \{\langle C_i, bond, C_{i+1}\rangle \mid 1 \leq i \leq 5\}$, $c_1 = \langle C_6, bond, C_1\rangle$, $P_2 = \{\langle N, bond, O_1\rangle, \langle N, bond, C\rangle\}$, and $c_2 = \langle N, bond, O_2\rangle$. Via a Chase process, a subgraph G_s matching P_1 and P_2 (red edges) is identified, and the missing bonds c_1 and c_2 (blue edges) are enforced, generating a graph $G_g \models \{\varphi_1, \varphi_2\}$ that satisfies both constraints. The enriched graph suggests a chemically consistent molecular structure, as required by the constraints. □

2.3 Witnesses for Graph Inference Queries

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

Witness graph. Given an inference query $Q = (M, v_t, G)$ over a test node v_t and a graph ML model M: $G \rightarrow \mathbf{H}_v^{(L)}$, 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. 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 $\mathbf{H}_{v_t}^{(3)}$. Removing any 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 [26], or be implicitly induced from a learned masked adjacency matrix [27].

3 GROUNDING WITNESSES WITH CONSTRAINTS

As remarked earlier, a witness alone may not be aligned to meaningful real-world structures as it may only be faithful to the model [40]. 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 refer to the edges in ΔE as *grounded edges*.

We remark that the symbol $\oplus$ means "extended" but not "union", *i.e.,* G_g may contain G_s, a set of new edges 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 indicates 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 $\mathsf{aln}(G_s, \varphi) = 1$.

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

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

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

An Axiomization analysis. We justify $F(\cdot)$ with an axiomization analysis below. For any inference query $Q(M, v_t, G)$, let G^*_s be the optimal witness that maximizes $F(G^*_s)$. $F(\cdot)$ has the following properties. (1) **Scale invariance**. G^*_s remains to maximizes F regardless how conc, aln and cov 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 [1]).

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, 42]. 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 [1]), 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 $P \cup c$.

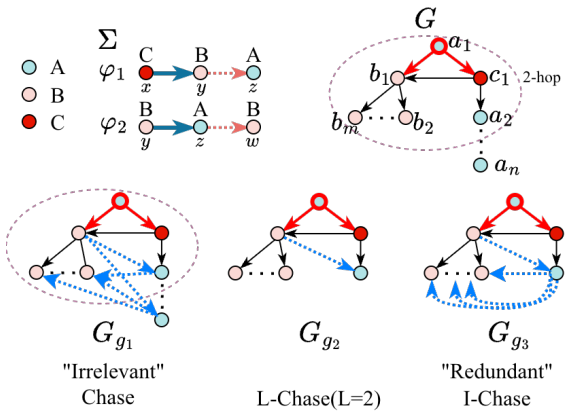


Figure 3: Generating grounded witnesses (Inf-Chase); unlike I-Chase as a fixpoint computation, L-Chase reasons with Σ up to L times from a set of "source" constraints Σ_s.

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.

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

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

(1) As P_1 is matched, a Chase, as a fixpoint process, enforces c_1 to insert edges between b_2 and any a node regardless of how large n is ($b_2 \rightarrow a_2, \ldots, a_n$). Most of edges fall outside the receptive field of v_t and thus become irrelevant (see G_{g_1}).

(2) In contrast, I-Chase restricts reasoning within $G^L(v_t)$, hence prevent a_i that is too far from a_1 to be considered for grounding. However, in the second round of chase, φ_2 is triggered, enforcing redundant, multiple semantically-equivalent structures (*e.g.,* $a_2 \rightarrow b_2, \ldots, b_m$), leading to an excessive number of "over-grounded" graphs that are all isomorphic.

(3) L-Chase strikes a balanced alternative: by performing up to just one more round of reasoning (as L=2, and G_s already include all hop-1 neighbors) within $G^L(v_t)$, it suffices to chase one round to produce an informative and grounded result (G_{g_2}) with less irrelevant or redundant edges. This illustrates the need to have a bounded hop, and bounded rounds of chase to avoid irrelevant and redundant provenance structure for grounding the witnesses. □

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 (φ'^{-1},I-Chase (φ, $G^L(v_t)$, 0), k), and the latter composite function is exactly computed by Inf-Chase in the two-phase Chase & Backchase process.

Algorithm 1 : 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.

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

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

Time cost. Both Chase and 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, 42], 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. 10, specifies Inf-Chase as an online generation process with (1) a further optimization of L-Chase with cost-effective and memoization structures, to reduce excessive inference and verification cost; and (2) ensures an "any-time" quality guarantee with a small delay time.

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

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 $\varphi_i = (P_i, c_i)$, *i.e.,* e indicates a common triple pattern in multiple

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

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

Figure 5: Procedure L-Chase

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

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

Figure 6: Procedure L-Backchase

constraints. (2) A shared map $\mathcal{M}$ with size $|\varphi| \times (p + 2)$, with p the size of the largest pattern P in 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.

Algorithm Outline. The main driving algorithm ApxIChase, and the procedures L-Chase and L-Backchase are illustrated in Figs. 10, 5 and 6, 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 [1]). The correctness and time cost follows from its counterpart of Inf-Chase.

We present the detailed analysis in [1].

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),

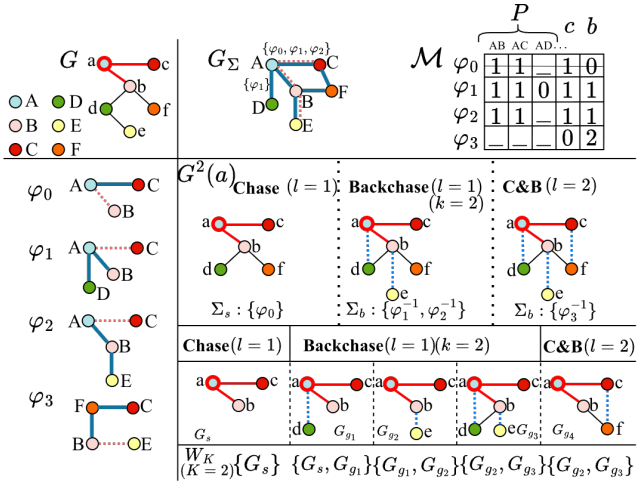


Figure 7: ApxIChase**: running of C& B in** 2 **rounds**

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.,* [19, 34] are readily applicable.

(2) To reduce cost from redundant inference computation, procedure VerifyWitness performs an incremental inference strategy. Instead of always re-start inference process from scratch (line 6 of VerifyWitness), it conducts a once-for-all inference for $Q(M, v_t, G^L(v_t))$ and caches the node embeddings. To verify if a candidate G_c is a witness, it directly loads the cached embeddings of any node v with $G^L(v) \subseteq G^L(v_t)$, and continue the message-passing from v to v_t, to incrementally update the node embeddings.

(3) Procedure L-Backchase reduces enforcement cost by monitoring $\mathcal{M}$ to optimize the enforcement cost. It maintains a lower bound of $|\Delta E|$, as the number of "missing edge matches", readily recorded as the number of 0 entries in $\mathcal{M}[i].P$, at any time. $\mathcal{M}$ also enables early suggestion and run-time pruning for 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 [1].

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

Example 8: Consider running ApxIChase on the example in Fig. 7. The original graph G (top-left) contains six typed nodes A–F, with four constraints $\Sigma = \{\varphi_0, \varphi_1, \varphi_2, \varphi_3\}$ shown at the left. Each constraint $\varphi_i = (P_i, c_i)$ is illustrated with its antecedent P_i (blue solid edges) and consequent c_i (pink dotted edge). The merged pattern G_Σ (top-right) encodes the union of $P_i \cup c_i$ for all $\varphi_i \in \Sigma$. The memoization map $\mathcal{M}$ records the grounding status of each constraint after one chase round. For example, $\mathcal{M}[0].b = 0$ (as all the pattern edges have a match without "0" entries) indicates that $G_s \models_0 \varphi_0$, hence φ_0 is satisfied and can be removed from subsequent enforcement.

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

(2) In the second round (L=2), constraint φ_3^{-1} is also triggered, enforcing the new edge (c, f). The bottom row shows the evolution of the top-K window W_K (with $K = 2$). Initially, G_s is discovered as a 0-grounded witness for φ_0. G_{g_1} and G_{g_2} represent $G_s \models_1 \varphi_1$ and $G_s \models_1 \varphi_2$, updating $W_K = \{G_{g_1}, G_{g_2}\}$. Upon the arrival of G_{g_3}, ApxIChase prefers G_{g_3} given its better coverage of both φ_1 and φ_2, replaces G_{g_1} with it. The current top-K sets contains G_{g_2}, G_{g_3}.

(3) The process continues, and after another round of Chase & Backchase, a new graph G_{g_4} satisfies φ_3 ($G_s \models_1 \varphi_3$), yet due to a weak link to a, it does not improve the F-score ranking, hence W_K remains unchanged. The algorithm returns W_K as a final result. □

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 [30]. 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 explainability analysis [27], 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 such as resistant distance [43], Shapley value [41], or sensitivity [22], can also be applied.

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

Outline. The algorithm follows the framework Inf-Chase with a L-Chase and a L-Backchase step to generate grounded witnesses. Unlike 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

Dataset	|V|	|E|	# node types	# attributes
BAShape [40]	2,020,000	12,055,704	4	1
Cora [30]	2708	5429	7	1433
ATLAS [2]	59,148	40,823	2	64
MUTAG [9]	17.9(avg.)	19.8(avg.)	7	7
ogbn-Papers100M [20]	111M	1.6B	40	128
Tree-Cycles [40]	67M	1.4B	5	5

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

arborescence. L-Backchase then applies Edmonds' algorithm [12] to derive G_g over the 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 Inf-Chase 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 [1]. 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

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

Datasets. We use six real-world graphs and one synthetic graph for evaluation (summarized in Table 1). (1) *MUTAG* [9], a set of 188 chemical compound graphs. Each graph has a class label indicating whether it is mutagenic or non-mutagenic. Node feature is a 7-dimensional one-hot encoding of atom types (*e.g.,* C, N, O). (2) *ATLAS* [2] is a cyber event graph for intrusion detection. Nodes represent system entities (e.g., process, file, connection), and edges denote data flow or dependencies. (3) *Cora* [30] is a citation network with 7 categories. Each node has a 1,433-dimensional feature vector, and edges represent citation links among papers. (4) *BAShape* [40] is a synthetic dataset generated by attaching house-shaped motifs to a Barabási–Albert base graph. A node label corresponds to structural roles (top, middle, bottom, or background). (5) *ogbn-Papers100M* [20] is a large-scale citation network with 1.6 billion citations (edges). Each paper has a 128-dimensional feature vector and is classified into one of 40 research fields. (6) *Tree-Cycles* [40] is a synthetic dataset with hierarchical graphs combining trees and cyclic structures, with 1.4 billion edges and nodes in five categories represented by 5-dimensional one-hot features.

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

Graph Models. We have pre-trained three classes of representative GNNs: GCNs [21], GATs [37], and GraphSAGE [16]. For each dataset, all methods are applied to the same set of GNNs to ensure comparability. Each GNN model is implemented using PyTorch Geometric, with 3 layers and a hidden dimension of 32. We use 70/15/15 split for train/val/test. Each model is trained for 200 epochs using Adam optimizer ($lr = 0.01$).

Algorithms. We implemented naiveChase, ApxIChase and HeuIChase, compared with two GNN explainers where subgraph structures can be derived for fair comparison. (1) *GNNExplainer* [40], an established post-hoc method that identifies subgraphs most influential to the model's prediction by optimizing a continuous edge mask through gradient-based learning. (2) *PGExplainer* [27], a learning-based method that learns a neural edge generator to produce explanations conditioned on node embeddings. Their generated subgraphs are treated as witnesses for comparison. In addition, to test the scalability, we also implemented paraIChase, a parallelized implementation of Inf-Chase in a shared memory, but by partitioning the inference query workload (see details in [1]).

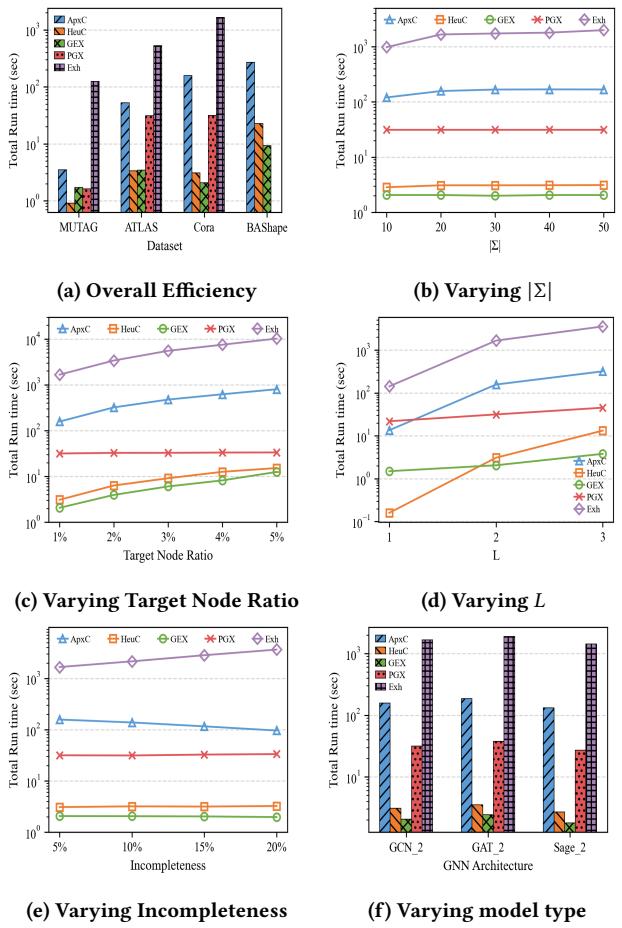
Evaluation Metrics. We report the efficiency and effectiveness of all the methods. (1) For Efficiency, for online algorithms (e.g., ApxIChase, HeuIChase) and naiveChase, we report the total response time including generation, verification, and window maintenance until no changes can be made; for baseline GNN explainers, we report the total runtime needed to generate the explanation subgraphs with the same environment, data, and task configuration. (2) For Effectiveness, we evaluate three metrics: *Coverage*, *Conciseness*, (both are as defined in Section 3), and *Fidelity* [41]. This measure quantifies how well a witness G_s preserves the model's output embeddings on G ("faithfulness")): $\text{fidelity}^-(G_s) = \Pr(Q(M, v_t, G)) - \Pr(Q(M, v_t, G_s))$, the smaller, the better.

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

7.2 Experimental Results

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

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



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

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

(e) Varying Incompleteness **(f) Varying model type**

Figure 8: Performance Evaluation: Efficiency

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

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

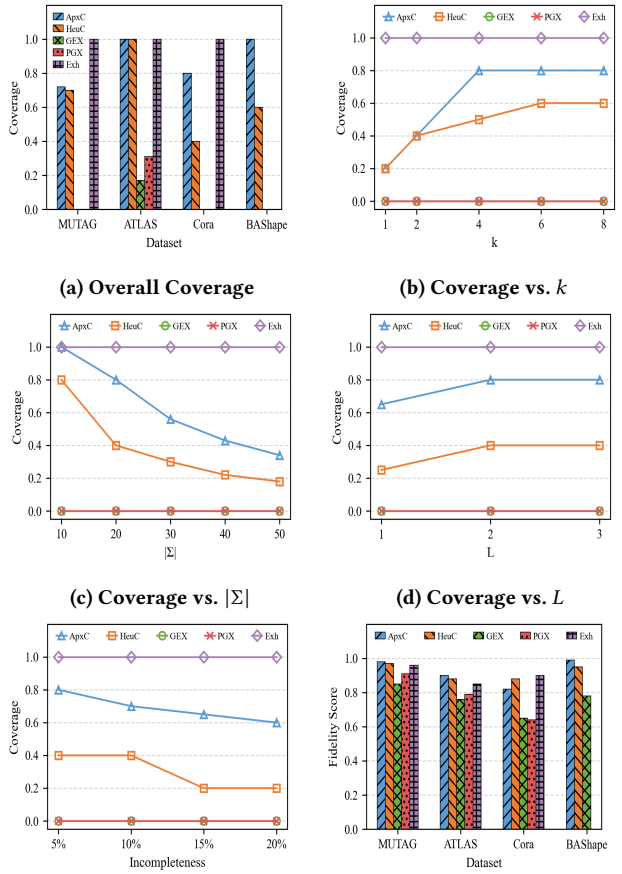
Varying the size of test set $|V_t|$. We vary the proportion of the target nodes from 1% to 5% on Cora while fixing other factors to their default values. As shown in Fig. 8(c), both ApxIChase and HeuIChase exhibit an approximately linear growth in total runtime as $|V_t|$ grows. This is expected, since each test node potentially triggers a separate round of verification and constraint-based grounding. Despite the growth, HeuIChase remains consistently "lightweight" – under 15 seconds even at 5% sampling – demonstrating its scalability and efficiency in grounding large inference query workload. In contrast, naiveChase incurs a cost that grows rapidly, due to the high cost of fixpoint enforcing process. GNNExplainer is more sensitive compared with PGExplainer as a posthoc approach, as the latter always performs global training over entire graphs.

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

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

Varying GNN *model type.* We examine the influence of backbone GNN architectures on runtime by evaluating all methods on Cora using three representative models: 2-layer GCN, GAT, and GraphSAGE. As shown in Fig. 8(f), our 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. We next evaluate the overall quality of witness generation in terms of conciseness, coverage, and fidelity. All the tests are conducted under the default configuration of (K=6, k=8, L=2, incompleteness=5%, $|\Sigma|$=20, $|V_t|/|V|$=1%), unless otherwise specified. Unless otherwise noted, we report the impact of individual factors on the Cora dataset.



(a) Overall Coverage **(b) Coverage vs. k**

(c) Coverage vs. $|\Sigma|$ **(d) Coverage vs. L**

(e) Coverage vs. Incompleteness **(f) Overall Fidelity**

Figure 9: Performance Evaluation: Effectiveness

Overall Coverage. We first compare the coverage over all the datasets under the default configuration, as shown in Fig. 9(a). naiveChase attains full coverage as it enforces all constraints before witness generation. Both 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 consider generating explanations that are grounded by constraints, yield near-zero coverage on most datasets. An exception occurs on ATLAS, where the underlying L-hop neighborhoods are mostly tree-structured, allowing them to be grounded by several constraints with tree patterns.

Coverage vs. budget k. We study how the constraint coverage changes with the budget $k \in \{1, 2, 4, 6, 8\}$ on Cora, while keeping other factors fixed. As shown in Fig. 9(b), the coverage of 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 v.s $|\Sigma|$. We vary the size of constraints $|\Sigma| \in \{10, 20, 30, 40, 50\}$ while fixing other factors at their default values. As shown in Fig. 9(c), both ApxIChase and HeuIChase exhibit a gradual decline in coverage as $|\Sigma|$ increases. This is expected, since under a fixed budget k, each method can only ground a limited number of constraints, making it harder to find grounded witnesses with high coverage. On the other hand, the trend is not very sensitive. We found that ApxIChase exploits G_Σ and M well to first reason with constraints that have high overlapping with chased ones, allowing the algorithms to maintain a reasonable coverage even for large $|\Sigma|$. GNNExplainer and PGExplainer remain unaffected, as they do not incorporate constraint grounding.

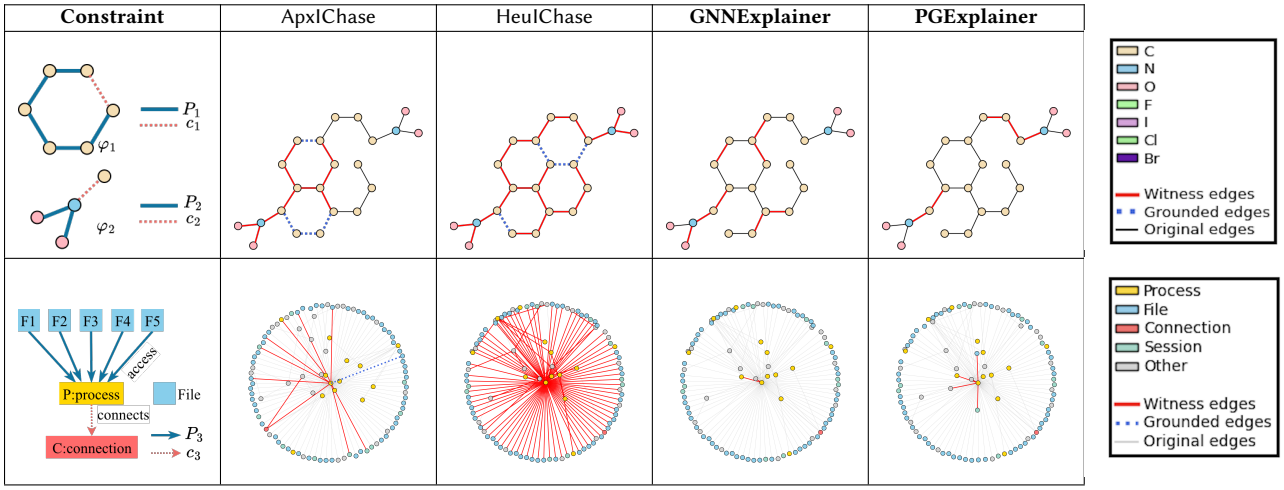
Coverage vs. L. We further analyze the effect of model depth (equivalently, the neighborhood hop count L) on constraint coverage, varying $L \in \{1, 2, 3\}$ while fixing other factors. As shown in Fig. 9(d), both ApxIChase and HeuIChase have coverage improved with larger L, as a wider receptive field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

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

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

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

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



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

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

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

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

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

Case II: Data Exfiltration Attack Investigation. We analyze a backdoor process flagged by a 3-layer GraphSAGE model for credential theft and data exfiltration. Table 2 (Row 2) compares reconstructed attacks under constraint $\varphi_3 : P_3 \rightarrow c_3$, requiring that any process accessing five sensitive files—F1:/etc/passwd, F2:.ssh/id_rsa, F3:cookies.db, F4:confidential.pdf, F5:credentials.txt—must also show outbound connection C:198.51.100.25:443. (1) 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$\rightarrow$F1–F5) and exfiltration (backdoor$\rightarrow$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 some unrelated files and processes. This may be due to its edge construction based on similarity. Yet as φ_3 holds, it can suggest structures to the downstream analysis to further isolate the true attack paths. (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 partial coverage but deviates from the constraint with connections to auxiliary sessions, missing critical files (F2, F3, F5). Its probabilistic sampling breaks the antecedent–consequent structure, yielding ungrounded witnesses.

8 CONCLUSION

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

REFERENCES

[1] 2025. Full version. https://github.com/anthonyche/Grounded_Witness/blob/main/Grounding_Graph_Inference_Analysis_with_Data_Constraints.pdf

[2] 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.

[3] 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.

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

[5] Peter Buneman, Sanjeev Khanna, and Tan Wang-Chiew. 2001. Why and where: A characterization of data provenance. Springer, 316–330.

[6] Marco Calautti, Mostafa Milani, and Andreas Pieris. 2023. Semi-Oblivious Chase Termination for Linear Existential Rules: An Experimental Study. *Proc. VLDB Endow.* 16, 11 (2023), 2858–2870.

[7] Ming Chen, Zhewei Wei, Bolin Ding, Yaliang Li, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. 2020. Scalable graph neural networks via bidirectional propagation. *Advances in neural information processing systems* 33 (2020), 14556–14566.

[8] James Cheney, Laura Chiticariu, Wang-Chiew Tan, et al. 2009. Provenance in databases: Why, how, and where. *Foundations and Trends® in Databases* (2009).

[9] Asim Kumar Debnath, Rosa L Lopez de Compadre, Gargi Debnath, Alan J Shusterman, and Corwin Hansch. 1991. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry* 34, 2 (1991), 786–797.

[10] Alin Deutsch, Lucian Popa, and Val Tannen. 2001. Chase & backchase: A method for query optimization with materialized views and integrity constraints. (2001).

[11] Jinlong Du, Senzhang Wang, Hao Miao, and Jiaqiang Zhang. 2021. Multi-Channel Pooling Graph Neural Networks.. In *IJCAI*. 1442–1448.

[12] Jack Edmonds et al. 1967. Optimum branchings. *Journal of Research of the national Bureau of Standards B* 71, 4 (1967), 233–240.

[13] Wenfei Fan and Ping Lu. 2019. Dependencies for graphs. *ACM Transactions on Database Systems (TODS)* 44, 2 (2019), 1–40.

[14] 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.

[15] 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.

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

[17] Myoungji Han, Hyunjoon Kim, Geonmo Gu, Kunsoo Park, and Wook-Shin Han. 2019. Efficient subgraph matching: Harmonizing dynamic programming, adaptive matching order, and failing set together. In *Proceedings of the 2019 international conference on management of data*. 1429–1446.

[18] Wook-Shin Han, Jinsoo Lee, and Jeong-Hoon Lee. 2013. Turboiso: towards ultrafast and robust subgraph isomorphism search in large graph databases. In *Proceedings of the 2013 ACM SIGMOD international conference on management of data*. 337–348.

[19] Yuxing Han, Ziniu Wu, Peizhi Wu, Rong Zhu, Jingyi Yang, Liang Wei Tan, Kai Zeng, Gao Cong, Yanzhao Qin, Andreas Pfadler, Zhengping Qian, Jingren Zhou, Jiangeng Li, and Bin Cui. 2022. Cardinality Estimation in DBMS: A Comprehensive Benchmark Evaluation. *Proceedings of the VLDB Endowment* 15, 4 (2022), 752–765. https://doi.org/10.14778/3503585.3503586

[20] 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).

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

[22] Douglas J Klein. 2010. Centrality measure in graphs. *Journal of mathematical chemistry* 47, 4 (2010), 1209–1223.

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

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

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] Daniel Mawhirter, Sam Reinehr, Connor Holmes, Tongping Liu, and Bo Wu. 2021. Graphzero: A high-performance subgraph matching system. *ACM SIGOPS Operating Systems Review* 55, 1 (2021), 21–37.

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

[31] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. 1978. An analysis of approximations for maximizing submodular set functions—I. *Mathematical programming* 14, 1 (1978), 265–294.

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

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

[34] Yufan Sheng, Xin Cao, Kaiqi Zhao, Yixiang Fang, Jianzhong Qi, Wenjie Zhang, and Christian S. Jensen. 2025. ACE: A Cardinality Estimator for Set-Valued Queries. *Proceedings of the VLDB Endowment* 18, 7 (2025), 2112–2125. https://doi.org/10.14778/3734839.3734848

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

[36] 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.

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

[38] 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.

[39] 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.

[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] 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).

[43] Mingzhe Zhu, Liwang Zhu, Huan Li, Wei Li, and Zhongzhi Zhang. 2024. Resistance distances in directed graphs: Definitions, properties, and applications. *Theoretical Computer Science* 1009 (2024), 114700.

[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: Formal Notations and Proofs

Graph Data Constraints: Specifications. The constraint set Σ contains a set of graph data constraints, where a data constraint $\varphi = (P, c)$ can be easily specified to the following types of constraints.

(1) A graph functional dependency (GFD) [15]. A GFD φ has a general form $(Q, X \rightarrow Y)$, where Q is a graph pattern query with a set of pattern nodes and edges, and each pattern node u is associated with a variable x_u from a finite set of variables U, and $X \rightarrow Y$ enforces a value binding constraint (such as a functional dependency) over the attribute set $\mathbf{A}$ of the nodes that match node variables in Q via subgraph isomorphism. A GFD φ' can be equivalently expressed by a data constraint $\varphi = (P, c)$, where P contains Q and enforces

an equivalent literal $x_u.X = x'_u.X$, over the two nodes in Q with variables involved in X, and c is a triple $(u, \emptyset, u')$ over the same pair of variables u and u', and enforces an equivalent literal $x_u.Y = x'_u.Y$. It expresses the semantics consistently with φ' as: "for any two nodes that match the nodes in pattern P, if they have the same values for X, they must have the same value on attributes Y".

(2) A graph association rule (GPAR) [14] is in the form of $P \rightarrow P'$, where P and P' are graph pattern queries, and the rule enforces the co-occurrence of the matches of P and P' in a graph G. A graph association rule γ can be expressed by a set of graph data constraints Σ, where each data constraint $\varphi = (P, c)$ has a same antecedent P with a shared set of nodes variables, and c is a distinct, single triple pattern (a pattern edge) of P' defined a set of shared variables. A graph $G \models \Sigma$ ensures the co-occurrence of any subgraph G_s that matches P and a corresponding subgraph induced by the edge matches of c conjunctively, over shared node variables.

(3) A graph generation dependency (GGDs) [35] has a form $(Q_s, \phi) \rightarrow (Q_t, \phi')$, defined on the node variables U, where ϕ and ϕ' are a set of differential constraints that specify equality or inequality value constraints over the attributes of nodes that match Q_s and Q_t, respectively. GGDs can be expressed by a set of data constraints Σ in a similar construction in (2), where each constraint $\varphi = (Q_s \cup Q_t), c$ requires the existence of a (possibly disconnected) subgraph that matches Q_s and Q_t simultaneously, and have the values of node matches satisfying ϕ and ϕ' as differential constraints.

(4) Similar, SHACL [32] is subsumed by a practical implementation of data constraints Σ, where a SHACL encodes a graph pattern P that resembles a SPARQL query, and c specifies a conjunction of value-binding constraints for RDF data validation.

In general, we consider graph data constraints as a type of tuple equality and generation constraints (TEDs and TGDs) specified for graph data. When c is , a constraint φ is a graph query (a base for a representative graph query languages such as SPARQL, Cypher, or Gremlin); otherwise, c is a unit constraint that enforces value binding to edges that match P, and a conjunction of triple patterns ensures the existence of a graph induced by such an edge matches whenever enforced as a data repairing rule.

Hardeness of k-bounded witness generation. We provide the following analysis to show the hardness of the witness generation problem. We first start with a hardness analysis for a witness verification problem. Given an inference query $Q = (M, v_t, G)$, a graph ML model M, and a subgraph G_s, it is to verify if G_s is a witness of the result of Q.

Lemma 4: *The witness verification problem is in* PTIME. □

Proof. This result can be verified by constructing a PTIME procedure that invokes the inference process of the graph ML model M, and checks by definition if G_s is either factual or counterfactual, in polynomial time. □

We next consider the k-grounded witness verification problem. Given an inference query $Q = (M, v_t, G)$, a graph ML model M, a graph G_g with a subgraph G_s of G and a set of grounded edges ΔE, a set of data constraints Σ, and an integer k, it is to verify if G_s is a minimal k-grounded witness of the result of Q.

Lemma 5: *The k-grounded witness verification is* coNP*-hard.* □

Proof. The hardness carries over from the hardness of the validation problem of graph functional dependencies (GFD), which is a special case of data constraints. Let G_s be a subgraph that passed the PTIME witness verification problem (see Lemma 4), by calling a GNN inference function. It then suffices to decide if $G_s \models_k \varphi$. Let $k = 0$, and φ is a single graph functional dependency. Then we consider a reduction from a GFD validation problem. The latter is to verify if $G_s \models \varphi$ as a GFD. As the validation problem for GFDs has been verified to be coNP-hard, hence the k-grounded witness verification remains coNP-hard. □

Hardness of Witness generation. We next prove Theorem 1.

Proof. The hardness can be shown by constructing a 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 under constraints 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 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 follows. □

Properties of I-Chase. We next provide proof to show the properties of I-Chase. We provide the detailed proof of Lemma 2. Before we present the formal proof, we introduce several notations below.

In an analogy of Chase that enforces data constraints to relational tables, we consider I-Chase an extended Chase process over graph G and Σ to "transform" G to a graph G' such that $G' \models \Sigma$. We formalize the computation of I-Chase below. A I-Chase is a rewriting process $< \mathcal{S}, \mathcal{E}, \mathcal{O} >$ that consists of the following.

(1) $\mathcal{S}$ is a set of states, and each state s is a pair (Σ', G_s) such that $\Sigma' \subseteq \Sigma$, and G_s is a subgraph of G.

(2) $\mathcal{O} \subseteq O_\Sigma \times O_G$ is a set of operators, where each operator o is a pair (o_Σ, o_G). Here an operator o_Σ is from a set $\{\oplus\varphi, \ominus\varphi$, for a constraint $\varphi \in \Sigma$; and $\oplus\varphi$ (resp. $\ominus\varphi$) is to "add" (resp. "remove") a constraint to (resp. from) a current subset $\Sigma' \subseteq \Sigma$. o_G is from a set $\{\oplus e, \ominus e\}$ for an edge $e \in E$ of G; which is either an edge insertion ($\oplus e$) or deletion ($\ominus e$) to be applied to a current subgraph $G' \subseteq G$. Both o_Σ and o_G can be $\emptyset$.

(3) $\epsilon_o \in \mathcal{E}$ is a transition between two states $s_1 = (\Sigma_1, G_{s_1})$ and $s_2 = (\Sigma_2, G_{s_2})$. A transition $\epsilon_o(s_1, s_2)$ applies an operator $o = (o_\Sigma, o_G)$ simultaneously applies o_Σ to Σ_1 to update it to Σ_2, and o_G to G_{s_1} to G_{s_2}, thus transits from state s_1 to s_2.

A state $s = (\Sigma_s, G_s)$ is a *terminating* state, if (1) G_s is a witness *w.r.t.* the input inference query Q, and (2) $G_s \models \Sigma_s$.

I-Chase *sequence*. A *sequence* ρ of I-Chase is a sequential, consecutive processing of transitions ($\rho = \{\epsilon_1, \ldots \epsilon_n\}$) from a starting state

s_1 to a *result* state s_n. A sequence ρ is *terminating*, if its result s_n is a terminating state.

Runs of I-Chase. A running of I-Chase starts from an initial state $s_0 = (\Sigma, G)$. The running spawns multiple transitions at a state s with applicable operators. The running is terminating, if it contains at least one terminating sequence.

We start with an intuitive explanation below.

Termination. The vertex set $N^L(v_t)$ is finite, bounding possible edges by $|N^L(v_t)|$. I-Chase adds edges monotonically (*i.e.,* $G^{(i)} \subseteq G^{(i+1)}$) and never delete edges. Since only finitely many edges can be added, the fixed-point computation must reach a saturated graph G' where no constraint $\varphi \in \Sigma$ is applicable. Hence, I-Chase terminates after a finite number of enforcement steps.

Church-Rosser. A rewriting system is Church-Rosser if all execution sequences converge to a common extension. We prove this via: (1) Local Confluence: For any graph G^i during execution, suppose $\varphi_1 = (P_1, c_1)$ and $\varphi_2 = (P_2, c_2)$ are applicable. Adding edge e_{c_1} does not affect the match of P_2(Σ contains only positive conditions, no deletions.) Applying both yields $G^{(i)} \oplus \{e_{c_1}, e_{c_2}\}$ regardless of order. Hence all critical pairs merge. (2) Global Confluence: I-Chase is terminating and locally confluent. By Newman's Lemma, it is globally confluent: and two maximally executions from $G^{(0)}$ produce isomorphic saturated graphs satisfying Σ. Confluence directly implies the Church-Rosser property.

We provide a more formal proof below. The general proof idea is to represent I-Chase over Σ and G into an equivalent Chase process, which the latter can equivalently simulate as a reasoning system over a finite relational database. (1) We represent G as a table with a universal schema that contains a source node ID, a target node ID, and all nodes and edge attributes. (2) We rewrite each constraint φ as either a TEG for any literal that enforces an equality condition in c, and a TGD, for any c with non-empty u_p, to enforce a new tuple in accordance with the insertion of a new edge in G. (3) The additional literals from P are treated as data constraints posed by the head of the TGDs or TEDs, defined on the node variables. We can then verify that the Church-Rosser property holds for the above Chase process, by reducing it to an equivalent, special case of the Chase process for graph dependencies [13], which has been shown to have Church-Rosser property.

Lemma 6: *The Church-Rosser Property does not hold for* L-Chase. □

Proof. Unlike I-Chase, which guarantees convergence to a unique fixed point regardless of the enforcement order, L-Chase imposes a strict limit of L reasoning rounds. This bound forces the procedure to terminate once L rounds are exhausted, potentially before saturation is achieved. Consequently, if multiple constraints are applicable, prioritizing different constraints will produce distinct graph states upon reaching round L. Since the procedure halts at this limit, it precludes the subsequent steps necessary to unify these states and achieve confluence. Thus, the output of L-Chase is order-dependent. □

Axiomatic Analysis. We show that the function $F(G_s)$ is scale-invariant, consistent, non-monotonic, and independent. We first show that for any given function $F(\cdot)$, there exists an optimal solution $\mathcal{G}_s$ that can maximize $F(\cdot)$. This readily follows as one can run an exhaustive search that invokes I-Chase process as a fixpoint computation, to enumerate the solutions and identify the one that maximizes $F(\cdot)$. As the computation I-Chase is terminating, (no edge will be repeatedly inserted once enforced in G via an I-Chase sequence, *i.e., G* is not a multigraph), there always exists an optimal solution G_s^*. We can readily verify the following properties.

Notation	Description
$G = (V, E)$	graph G, V: the set of nodes, E: the set of edges
$\|G\|$	size of G; $\|G\| = \|E\|$
$N^L(v)$	nodes in the L-hop of v
$G^L(v)$	L-hop subgraph of v
M	a GNN model
v_t, V_T	a test node, a set of test nodes
$Q = (M, v_t, G)$	an inference query
$\mathcal{Q} = (M, V_T, G)$	a query workload
P, c	antecedent, consequent
$\varphi = (P, c)$	a data constraint
Σ	a constraint set
G_s	a witness graph
G_g	a grounded provenance graph

Table 3: Summary of Notations.

(1) Scale-free property claims that $F(\cdot)$ is insensitive to the scaling of the weighted terms that capture user preference. That is, for any $F'(\cdot) = c * F(\cdot) = \alpha * c \cdot \mathsf{conc}(\cdot) + \beta * c \cdot \mathsf{aln}(G_s) + \gamma * c \cdot \mathsf{cov}(G_s)$, *i.e.,* all terms are scaled up by a scalar c, if $G_s^* = \arg\max F(G_s)$, for all witnesses G_s, then $G_s^* = \arg\max F'(G_s)$. This is easy to verify by the existence of G_s^* and a proof of contradiction.

(2) Non-monotonicity states that for two witnesses G_s and G_s' with $|G_s| \geq |G_s'|$, it is possible that $F(G_s) \geq F(G_s')$ or $F(G_s) \leq F(G_s')$. It suffices to consider a case where $\gamma = \alpha = 0$, *i.e.,* only alignment cost is considered ($F(\cdot) = 1 - \frac{\delta E}{k}$. Assume $|G_s| \geq |G_s'|$, both of the following cases are possible: (a) G_s' is a 0-grounded witness with $\mathsf{aln}(G_s') = 1$, while G_s, although having more edges still require $|\Delta E| > 0$ edges to be grounded. (b) G_s is 0-grounded, and G_s' is a subgraph of G_s that lacks some edges to be grounded, hence $\mathsf{aln}(G_s') < \mathsf{aln}(G_s)$. This alone verifies the non-monotonicity of $F(\cdot)$.

(3) Independence. This can be verified by the following. (a) G_s alone determines the score of conc and cov. (b) Given k, any witness G_s with its current edge set uniquely specifies a finite set of all possible provenance graphs G_b via finite, deterministic I-Chase sequences, each enforces at most k edges. The alignment cost can thus be determined by verifying the subgraphs induced by the nodes from a bounded set $N^{k+L}(v_t)$, independent of the rest of the graph G.

9.2 Appendix B: Algorithms and Analysis

Implementation of naiveChase. The algorithm naiveChase enforce each graph constraint $\varphi \in \Sigma$ in $G^L(v_t)$.

(1) The I-Chase procedure extends Chase algorithm to perform a fixpoint enforcement of Σ as follows. Whenever there is an applicable constraint $\varphi = (P, c)$, it iteratively retrieves a subgraph $G_\varphi = G_s \oplus e_c$, where G_s is a subgraph of G that matches P (obtained by invoking fast subgraph match enumeration algorithms *e.g.,* [17, 18, 29]), and

Algorithm 4 : **Procedure** UpdateWK (W_K, $\mathcal{G}_g$, K, k,)

```
1: for G_g ∈ 𝒢_g do
2:     compute F(G_g)                                  ▷ F(G_g) = F(G_s)
3:     if |W_K| < K then
4:         heappush(W_K, (G_g, F(G_g))) else
5:         let (G_g^min, F_min) be the heap-min of W_K
6:         if F(G_g) > F_min then
7:             heappushpop(W_K, (G_g, F(G_g)))
8: return W_K
```

Figure 10: Procedure UpdateWK

e_c is a single edge that matches the consequent (triple pattern) c, either inserted (enforced), or originally exists in G.

(2) For each such subgraph $G_\varphi = G_s \oplus e_c$, it invokes a verification process VerifyWitness, which performs the inference of M over G_φ to test whether G_φ is a witness for the inference query $Q(M, v_t, G)$. If so, it adds 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 aggregated quality score.

Procedure UpdateWK (line 14, Algorithm ApxIChase**; Section 6)**. Given $\mathcal{G}_g$ obtained from the backchase phase (line 12,ApxIChase), UpdateWK iterates over each grounded provenance graph $G_g \in \mathcal{G}_g$, evaluates the quality score $F(G_g)$ (here $F(G_g) = F(G_s)$ for the witness $G_s \subseteq G_g$, for presentation convenience) based on conciseness, alignment cost, and coverage, and maintains a window W_K by greedily selecting the top-K witnesses with the highest scores(*i.e.,* the highest value of $F(G_g)$). Whenever $F(G_g)$ exceeds the minimum score in W_K, the corresponding entry $(G_g, F(G_g))$ replaces the lowest-ranked witnesses with the current best choice. This ensures that W_K always retains the K grounded graphs whose contained witnesses collectively maximize $\sum_{G_g \in W_K} F(G_g)$.

Optimization: Pruning Σ. To reduce redundant constraint enforcement, ApxIChase applies a structural pruning rule over Σ to avoid repeatedly checking constraints whose antecedents are subsumed by others.

Lemma 7: *Given two constraints $\varphi_i = (P_i, c)$, and $\varphi_j = (P_j, c)$, where P_i is a subgraph of P_j. If $G_s \models_k \varphi_i$, then $G_s \models \varphi_j$.* □

That is, whenever G_s is k-grounded by φ_i, it is ensured that G_s is k-grounded by φ_j. Hence φ_j can be safely pruned.

Proof sketch: The idea of our proof is to specify a case of logic implication between φ_i and φ_j, following a counterpart of the sound and complete rule inference system of graph entity dependencies [13]. Consider two constraints $\varphi_i = (P_i, c)$ and $\varphi_j = (P_j, c)$ with $P_j \supseteq P_i$ (*i.e.,* P_j is a subgraph of P_i). Let ΔE_i and ΔE_j denote the minimal sets of added edges required to make G_s k-grounded by φ_i and φ_j, respectively. Since $P_j \subseteq P_i$, it follows that $\Delta E_j \subseteq \Delta E_i$ and $|\Delta E_j| \leq |\Delta E_i| \leq k$. By definition of k-groundedness, if $|\Delta E_i| \leq k$, then $G_s \models_k \varphi_i$, and $G_s \models_k \varphi_j$ by adding only a subset ΔE_j of edges. Thus, any graph $G_g = G_s \oplus \Delta E_i$ that satisfies φ_i (*i.e.,* $G_s \models_k \varphi_i$) already ensures that $G_g \models \varphi_j$ (*i.e.,* $G_s \models_k \varphi_i$), without the need to further identify ΔE_j. □

Analysis of ApxIChase. We prove that ApxIChase terminates and returns K feasible k-grounded witnesses as an approximation for the optimal counterpart that can be retrieved from the verified witnesses at any time.

Correctness. The algorithm executes at most L rounds(line 3). In each round $l \in [1, L]$, L-Chase (line 6) performs bounded pattern matching over $G^L(v_t)$ without edge insertion, terminating when all source constraints Σ_s are processed. L-Backchase (line 12) enforces at most k new edges per applicable inverse constraint $\varphi^{-1} \in \Sigma_b$, as tracked by budget $\mathcal{M}[i].b$(Figure 6, line 3), since $G^L(v_t)$ is finite and each edge insertion reduces the available budget, the backchase phase terminates after at most $|\Sigma| \cdot k$ edge insertions. The outer loop completes in bounded steps and guarantees termination.

Each subgraph admitted to W_K satisfies two conditions: (1) VerifyWitness ensures that every candidate G_c is either factual or counterfactual, (2) In L-Backchase, for each $\varphi^{-1} = (c, P)$, the algorithm inserts a minimal edge set ΔE to enforce at least one match of P, making the witness G_s satisfy $G_s \models_k \varphi$. The memoization map $\mathcal{M}$ records lower bounds on $|\Delta E|$, ensuring minimality. Consequently, $\mathcal{G}_g$ is a set of k-grounded minimal witnesses.

Approximability. UpdateWK implements a greedy replacement strategy: upon generating a new G_g, it evaluates the marginal gain in quality score $F(\cdot)$(Section 3) and replaces the lowest-scoring witness in W_K if $F(G_g)$ improves the aggregated objective. Following the standard analysis of monotone submodular maximization with cardinality constraint K, this greedy maintenance achieves a $(1 - 1/e)$- approximation of the optimal K-witnesses set at any request time, formalized in Theorem 3. The approximation guarantee follows because the quality function $F(\mathcal{G}_g) = \sum_{G_g \in \mathcal{G}_g} F(G_g)$ is monotone submodular:

Proof. For any subset $\mathcal{A} \subseteq \mathcal{G}_g$ we define

$$F(\mathcal{A}) = \sum_{G_g \in \mathcal{A}} F(G_g).$$

Non-negativity. By definition of $F(\cdot)$, for every witness G_s we have $0 \leq \text{conc}(G_s), \text{aln}(G_s), \text{cov}(G_s) \leq 1$ and $\alpha, \beta, \gamma \geq 0$. Hence $F(G_g) = F(G_s) \geq 0$ for all $G_g \in \mathcal{G}_g$, and therefore $F(\mathcal{A}) = \sum_{G_g \in \mathcal{A}} F(G_g) \geq 0$ for any $\mathcal{A} \subseteq \mathcal{G}_g$.

Monotonicity. Let $\mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{G}_g$. Then

$$F(\mathcal{B}) - F(\mathcal{A}) = \sum_{G_g \in \mathcal{B} \setminus \mathcal{A}} F(G_g) \geq 0,$$

since each $F(G_g)$ is non-negative. Thus $F(\mathcal{A}) \leq F(\mathcal{B})$, and $F(\cdot)$ is monotone as a set function over grounded witnesses.

Submodularity. Let $\mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{G}_g$ and $G'_g \notin \mathcal{B}$. The marginal gain of adding G'_g to $\mathcal{A}$ is

$$F(\mathcal{A} \cup \{G'_g\}) - F(\mathcal{A}) = \Big(\sum_{G_g \in \mathcal{A}} F(G_g) + F(G'_g) \Big) - \sum_{G_g \in \mathcal{A}} F(G_g) = F(G'_g),$$

and the marginal gain of adding G'_g to $\mathcal{B}$ is

$$F(\mathcal{B}\cup\{G'_g\})-F(\mathcal{B}) = \Big(\sum_{G_g\in\mathcal{B}} F(G_g)+F(G'_g)\Big)-\sum_{G_g\in\mathcal{B}} F(G_g) = F(G'_g).$$

Thus the marginal gain of G'_g is the same for $\mathcal{A}$ and $\mathcal{B}$ and, in particular, it does not increase as the set grows. Hence $F(\cdot)$ satisfies the diminishing-return condition and is submodular. As $F(\mathcal{A})$ aggregates the sum of fixed singleton weights $F(G_g)$, $F(\cdot)$ is a modular function, which is a special case of submodular functions. □

Remark. It is important to distinguish the two uses of $F(\cdot)$ in our analysis. In Axiomatic Analysis, $F(G_s)$ is defined on the node/edge set of a *single* grounded witness G_s. There we showed that $F(G_s)$ is scale-invariant, consistent, and independent, but *non-monotonic* with respect to graph size: given two witnesses G_s and G'_s with $|G_s| \geq |G'_s|$, it is possible that $F(G_s) \geq F(G'_s)$ or $F(G_s) \leq F(G'_s)$, as demonstrated by the alignment-only examples in our axiomatic analysis. In contrast, in the approximability analysis above, $F(\mathcal{A})$ is a *graph-set function* whose argument is a set $\mathcal{A} \subseteq \mathcal{G}_g$ of grounded witnesses, and whose value aggregates the per-witness scores $F(G_g)$ over all $G_g \in \mathcal{A}$. By construction, this set function is non-negative, monotone, and (modular) submodular over grounded witnesses. Thus, the monotonicity and submodularity of $F(\mathcal{A})$ as a function over *witness sets* do not contradict the non-monotonicity of $F(G_s)$ as a function over the node/edge set of a single witness graph. By [31], greedy maximization of such a monotone submodular function under cardinality constraint K yields a $(1-1/e)$-approximation, in our modular setting, the greedy maintenance of W_K in UpdateWK also ensure the approximation guarantee.

Cost Analysis. The delay time per request is $O(|\Sigma|^2 \cdot |G^L(v_t)| + L|\Sigma|^2+|\Sigma|K^2)$. The dominant term $|\Sigma|^2|G^L(v_t)|$ arises from a double loop: L-Chase generates $O(|\Sigma||G^L(v_t)|)$ candidate witnesses, each of which is checked against all $|\Sigma|$ constraints to build Σ_b via the memoization map $\mathcal{M}$. The $L|\Sigma|^2$ term captures L-round constraint propagation and pairwise interaction during L-Backchase, where enforcing φ^{-1} triggers cascading updates to $\mathcal{M}[i].b$. The $|\Sigma|K^2$ term reflects UpdateWK's greedy replacement: each of the $O(|\Sigma|K)$ grounded graphs G_g is evaluated against the current K witnesses, costing $O(K)$ per comparison.

Analysis of HeuIChase. We present the detailed analysis below.

Correctness. The L-Chase correctly produce factual or counterfactual witnesses via tree matching algorithms and incremental verification, and L-Backchase enforces k-groundedness via inverse constraints, following their correctness analysis as in ApxIChase. It invokes Edmonds' algorithm to induce a maximum-weight arborescence G_g on the graph compressed by contracting G_s into a supernode. The contraction ensures that $G_s \subseteq G_g$, while Edmonds' algorithm ensures that G_g satisfies Σ with minimal ΔE, yielding k-grounded witnesses as trees.

Cost Analysis. The complexity reduction stems from two substitutions that avoid exhaustive subgraph enumeration.

(1) Tree Pattern Matching: Matching a tree pattern P of size p against m sampled trees costs $O(m|G^L(v_t)| + mp^2)$. The first term accounts for sampling m spanning trees (via linear-time random walk), and the second term mp^2 reflects tree-isomorphism checks: each of the m trees is matched against P in $O(p^2)$ time by aligning root-to-leaf paths. Compared to ApxIChase's $O(|G^L(v_t)|^p)$ subgraph enumeration, this is exponentially faster when p is bounded. Since p is bounded by a small constant, $O(mp^2)$ reduces to $O(m)$ per constraint, with a matching cost in $O(L\cdot m\cdot|\Sigma|)$. The $O(m|G^L(v_t)|)$ sampling cost is a one-time overhead per round.

(2) Weighted Arborescence Computation: The low cost rests on *node contraction*: shrinking G_s into a supernode s is safe because all edges in G_s already satisfy antecedent P of its generating φ. The remaining problem—connecting s to the rest of $G^L(v_t)$ while enforcing inverse constraints $\varphi^{-1} \in \Sigma_b$—reduces to finding a minimum-cost arborescence rooted at s. *Edmonds' algorithm* with Fibonacci heap support solves this in $O(|G^L(v_t)|\log|G^L(v_t)|)$ time per round [12]. Edge weights are derived from embedding similarity to ensure that the resulting arborescence preserves inference fidelity. Over L rounds, the total arborescence cost is $O(L|G^L(v_t)|\log|G^L(v_t)|)$. Summing both components yields

$$O(L\cdot m\cdot|\Sigma| \;+\; L|G^L(v_t)|\log|G^L(v_t)|).$$

The first term dominates on sparse, tree-like graphs (*e.g.,* citation networks), while the second term dominates on dense neighborhoods. In practice, m is a small constant (*e.g.,* $m = 5$ in our experiments) to balance witness diversity and runtime, making HeuIChase 10–100× faster than ApxIChase on hierarchical datasets, as shown in Figure 8.

Algorithm 5 : paraIChase: Parallel Witness Generation

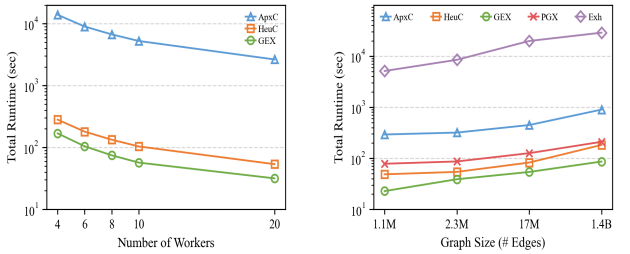
Input: Graph G, constraint set Σ, model M, target node set V_T, integers L,K,k, #workers w, ALG ∈ {ApxC, HeuC, Exh, GEX, PGX}
Output: $\{(v, W_K(v))|v \in V_T\}$

```
Coordinator computes G^L(V_T) = {(v_t, G^L(v_t))|v_t ∈ V_T}
P ← LoadBalance(G^L(V_T), w)    ▷ load balance by |G^L(v_t)|
for i = 1 to w do in parallel
    W_K^(i) ← ∅
    for all (v_t, G^L(v_t)) ∈ P_i do
        Execute ALG
        W_K^i ← W_K^(i) ∪ {(v_t, W_K(v_t))}
return ⋃_{i=1}^{w} W_K^(i)
```

Figure 11: Algorithm paraIChase**: Parallelized witness generation framework with load-balanced subgraph distribution.**

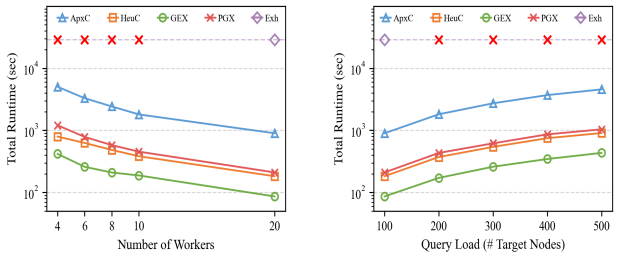
9.3 Appendix C: Complementary Experimental Study and Discussion

Parallel Algorithm. Algorithm 11 presents paraIChase, a parallel witness generation framework. A coordinator process induces L-hop subgraphs $\{(v_t, G^L(v_t))|v_t \in V_T\}$ and partitions them across w workers via First-Fit Decreasing bin-packing based on subgraph size $|G^L(v_t)|$, using a min-heap over worker loads to mitigate stragglers. Each worker independently runs one of the base algorithms ALG ∈ {ApxC, HeuC, Exh, GEX, PGX} on its assigned subgraphs. Global parameters are shared; workers access subgraphs via indexed slicing to avoid duplication. Verification is performed in mini-batches of



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

(b) Runtime Varying Graph Size on Tree-Cycles



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

(d) Runtime Varying Query Load on Tree-Cycles

Figure 12: Scalability Test

size 32 per worker to amortize inference costs. The coordinator aggregates per-target results $W_K(v_t)$ to compute final metrics.

Exp-3: Scalability. We report the scalability of our parallel framework (paraIChase) using both real and synthetic graphs. In all the tests, L=2, k=8, K=6, and we use 100 target nodes by default.

OGBN-Papers100M. Figure 12(a) shows that both ApxIChase and HeuIChase achieve clear speedups when scaling from 4 to 20 workers. Despite highly unbalanced subgraph sizes (some >80K edges), runtimes remain tolerable—dropping from 13.9K→2.6K s for ApxIChase and 282 → 54 s for HeuIChase. PGExplainer and naiveChase fail to scale, while GNNExplainer remains fastest but saturates beyond 10 workers.

Tree-Cycles. We further evaluate scalability on the synthetic Tree-Cycles dataset under three settings. (1)*Varying graph size:* As the graph grows from 1.1M to 1.4B edges (≈25 GB), both ApxIChase and HeuIChase show moderate runtime growth and remain tractable, while naiveChase exceeds 28,800 s (>8 h) with 20 workers and causes severe memory inflation. Each processor handles 4–6 L-hop subgraphs (≈4.6K edges). Under naiveChase, enforcing all constraints can double or triple subgraph size, peaking at ≈1 GB per processor (≈20 GB total). In contrast, ApxIChase and HeuIChase only operate on compact witnesses—typically 20–40% of subgraph size—keeping per-processor memory below 200–300 MB. Overall, they reduce memory by 4–5× versus naiveChase while maintaining stable runtime and balanced load. (2)*Varying processor count:* Fixing the 1.4B-edge graph and 100 target nodes, we vary workers from 4 to 20 (Fig. 12(c)). Both ApxIChase and HeuIChase achieve sublinear yet consistent speedups, limited mainly by coordination and I/O during extraction and aggregation. The makespan decreases by over 5×, showing effective load balance and low communication contention under high parallelism. (3)*Varying query load:* On the same graph with 20 workers, increasing target nodes from 100 to 500 (Fig. 12(d)) yields nearly linear runtime growth, dominated by extraction and grounding, with consistent trends across algorithms. Overall, our methods achieve 1–2 orders of magnitude runtime improvement and 4–5× lower memory than the exhaustive naiveChase, maintaining stable scalability across graph size, worker count, and query load.