Approximate Summaries for Why and Why-not Provenance (extended version)

Seokki Lee, Bertram Ludäscher, Boris Glavic

IIT DB Group Technical Report IIT/CS-DB-2019-01

2019-12

http://www.cs.iit.edu/~dbgroup/

Approximate Summaries for Why and Why-not Provenance (Extended Version)

Seokki Lee
Illinois Institute of Technology
slee195@hawk.iit.edu

Bertram Ludäscher
University of Illinois, Urbana-Champaign
ludaesch@illinois.edu

Boris Glavic
Illinois Institute of Technology
bglavic@iit.edu

ABSTRACT

Why and why-not provenance have been studied extensively in recent years. However, why-not provenance and—to a lesser degree—why provenance, can be very large resulting in severe scalability and usability challenges. In this paper, we introduce a novel *approximate summarization* technique for provenance which overcomes these challenges. Our approach uses patterns to encode (why-not) provenance concisely. We develop techniques for efficiently computing provenance summaries balancing informativeness, conciseness, and completeness. To achieve scalability, we integrate sampling techniques into provenance capture and summarization. Our approach is the first to scale to large datasets and to generate comprehensive and meaningful summaries.

PVLDB Reference Format:
Seokki Lee, Bertram Ludäscher, Boris Glavic. Approximate Summaries for Why and Why-not Provenance. *PVLDB*, 12(xxx): xxxx-yyyy, 2019.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

1. INTRODUCTION

Provenance for relational queries [11] explains how results of a query are derived from the query's inputs. In contrast, why-not provenance explains why a query result is missing. In prior work, we have shown that why and why-not provenance can be treated uniformly as provenance for first-order (FO) queries (= non-recursive Datalog with negation) [17] and have implemented this idea in a system called *PUG (Provenance Unification through Graphs)* [20, 21]. *Instance-based* [12] why-not provenance techniques determine which existing and missing data from a query's input is responsible for the failure to derive a missing answer of interest. These techniques either (i) enumerate all potential ways of deriving a result (*all-derivations* approach) or (ii) return only one possible, but failed, derivation or parts thereof (*single-derivation* approach). For instance, Artemis [13], Huang et al. [14], and PUG [22, 20, 21] are all-derivations approaches while the Y! system [34, 33] is a single-derivation approach.

Proceedings of the VLDB Endowment, Vol. 12, No. xxx
ISSN 2150-8097.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

$$r_1 : \texttt{AL}(N, R) :- \texttt{L}(I, N, T, R, \textsf{queen anne}, E), \texttt{A}(I, \textsf{2016-11-09}, P)$$

Listing (input)

Id	Name	Ptype	Rtype	NGroup	Neighbor
8403	cp: central place	apt	shared	queen anne	east
9211	plum	apt	entire	ballard	adams
2445	cozy homebase	house	private	queen anne	west
8575	near SpaceNeedle	apt	shared	queen anne	lower
4947	seattle couch	condo	shared	downtown	first hill
2332	modern view	house	entire	queen anne	west

Availability (input)

Id	Date	Price
9211	2016-11-09	130
2445	2016-11-09	45
2332	2016-11-09	350
4947	2016-11-10	40

Available**L**istings (output)

Name	Rtype
cozy homebase	private
modern view	entire

Attribute	Id	Name	Ptype	Rtype	NGroup	Neighbor	Date	Price
#Distinct Values	6	6	3	3	3	5	2	4

Figure 1: Example Airbnb database and query

EXAMPLE 1. *Fig. 1 shows a sample of a real-world dataset recording AirBnB (bed and breakfast) listings and their availability. For each* `Listing`*, we record an* id, name, property type (`Ptype`)*,* room type (`Rtype`)*, its* neighborhood (`Neighbor`) *and* neighborhood group (`NGroup`)*. Neighborhood groups are larger areas including multiple neighborhoods. Relation* `Availability` *stores* ids *of listings with available* dates *and a* price *for each date. We refer to this sample dataset as* S-Airbnb *and the full dataset as* F-Airbnb.[1] *Bob, an analyst at AirBnB, is tasked to investigate a customer complaint about the lack of availability of shared rooms on* 2016-11-09 *in Queen Anne (*`NGroup` = queen anne*). Bob first uses Datalog rule* r_1 *from Fig. 1 to return all listings (names and room types) available on that date in Queen Anne. The query result confirms the customer's complaint, since none of the available listings are* shared *rooms. Bob now needs to investigate what led to this missing result.*

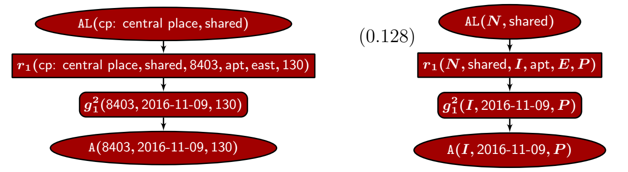
We refer to such questions as *provenance questions.* A provenance question is a tuple with constants and placeholders (upper-case letters) which specify a set of (missing) answers the user is interested in (all answers that agree with the provenance question on constant values). For example, Bob's question can be written as $\texttt{AL}(N, \textsf{shared})$. Allderivations approaches like PUG explain the absence of shared rooms by enumerating all derivations of missing answers that *match* Bob's question. That is, all possible bindings of the variables of the rule r_1 to values from the active domain (the values that exist in the database following the closed-world assumption) such that R is bound to shared and the tuple produced

[1] https://www.kaggle.com/airbnb/seattle

by the grounded rule is missing. While this explains why shared rooms are unavailable (any tuple with $R =$ shared), the number of possible bindings can be prohibitively large. Consider our toy example *S-Airbnb* dataset. Let us assume that only values from the active domain of each attribute are considered for a variable bound to this attribute to avoid nonsensical derivations, e.g., binding prices to names. Note that the number of distinct values per attribute are shown on the bottom of Fig. 1. Under this assumption, there are $6 \cdot 6 \cdot 3 \cdot 5 \cdot 4 = 2160$ possible ways to derive missing results matching $\texttt{AL}(N, \textsf{shared})$. For the full dataset *F-Airbnb* there are $\sim 15 \cdot 10^{12}$ possible derivations. Note that we use set semantics here (duplicates are not considered).

EXAMPLE 2. *Continuing with Ex. 1, assume that Bob uses PUG [20] (the system implementing our graph-based provenance model for FO queries) to compute an explanation for the missing result* $\texttt{AL}(N, \textsf{shared})$*. An example provenance graph fragment is shown in Fig. 2a. This type of provenance graph connects rule derivations (square nodes) with the tuples (oval nodes) they are deriving, rule derivation to the goals in their body (square nodes with rounded edges), and goals to the tuples that justify their success/-failure. Nodes are colored red/green to indicate failure/-success (goal and rule nodes) or absence/existence (tuple nodes). For S-Airbnb, the graph produced by PUG consists of all* 2160 *failed derivations of missing answers that match* $\texttt{AL}(N, \textsf{shared})$*. The fragment shown in Fig. 2a encodes one of these derivations: the shared room of the existing listing Central Place (*`Id` *8403) is not available on 2016-11-09 at a price of* \$130 *(and, thus, this derivation of the result fails), because the tuple (8403, 2016-11-09, 130) does not exist in the relation* `Availability` *(the second goal failed).*

Single-derivation approaches address the scalability issue of why-not provenance by only returning a single derivation or parts thereof. However, this comes at the cost of completeness. For instance, a single-derivation approach may return the derivation shown as a provenance graph fragment in Fig. 2a. However, such an explanation is not sufficient for Bob's investigation. What about other prices for the same listing? What about other listings? Do other listings from this area have shared rooms, but are not available at this date, or do they simply not have shared rooms? A single derivation approach cannot answer such questions since it only provides a single out of a vast number of failed derivations (or even only a sufficient reason for a derivation to fail as in [34, 33]), but misses many other equally valid explanations. For the S-Airbnb dataset, no shared rooms are available in Queen Anne on Nov 9th, 2016 because: 1) all the existing shared rooms of apartments (listings 8403 and 8575) in Queen Anne are not available on the requested date and 2) no listings in the West Queen Anne neighborhood (listings 2445 and 2332) have shared rooms. **Thus, returning only one derivation is insufficient for justifying the missing answer because only the collective failure of all possible derivations explains the missing answer.** Suppose that Bob has to explain the result of his investigation to his manager and is expected to propose possible ways of how to improve the availability of rooms in Queen Anne. His manager is unlikely to accept an explanation of the form *"There are no shared rooms available on this date, because listing 8403 is not available for \$130 on this day."*



(a) Partial provenance graph **(b) Provenance summary**

Figure 2: Explanations for the missing results $\texttt{AL}(N, \textsf{shared})$

Summarizing Provenance. In this paper, we present a novel approach that overcomes the drawbacks of both approaches. Specifically, we efficiently create summaries that compactly represent large amounts of provenance information. We focus on the algorithmic and formal foundation of this method as well as its experimental evaluation (we demonstrated a GUI frontend in [21] and our vision in [23]).

EXAMPLE 3. *Our summarization approach encodes sets of nodes from a provenance graph using* "pattern nodes"*, i.e., nodes with placeholders.*[2] *A possible provenance summary for* $\texttt{AL}(N, \textsf{shared})$ *is shown in Fig. 2b. The graph contains a rule pattern node* $r_1(N, \textsf{shared}, I, \textsf{apt}, E, P)$ *where* N*,* I*,* E*, and* P *are placeholders. For each such rule pattern node, our approach reports the amount of provenance covered by the pattern (shown to the left-top of the pattern node). This summary provides useful information to Bob: all the shared rooms of apartments in Queen Anne are not available at any price on Nov 9th, 2016 (their ids are not in relation* `Availability`*). Over F-Airbnb* $\sim 12.8\%$ *of derivations for* $\texttt{AL}(N, \textsf{shared})$ *follow this pattern.*

The type of patterns we are using here can also be modeled as selection queries and has been used to summarize provenance [32, 26] and for explanations in general [9, 10].

Selecting Meaningful Summaries. The provenance of a (missing) answer can be summarized in many possible ways. Ideally, we want *provenance summaries* to be *concise* (small provenance graphs), *complete* (covering full provenance), and *informative* (providing new insights). However, finding a solution that optimizes all three aspects is typically not possible. Consider two extreme cases: (i) any provenance graph is also a provenance summary (one without placeholders). Provenance graphs are complete and informative, but not concise; (ii) at the other end of the spectrum, an arbitrary number of derivations of a rule r can be represented a single pattern with only placeholders resulting in a maximally concise summary. However, such a summary is not informative since it only contains placeholders. We design a summarization algorithm that ranks patterns using a combination of completeness and informativeness and returns the top-k patterns wrt. this ranking (guaranteeing conciseness through k). The rationale behind optimizing for both completeness and informativeness is to ensure that summaries are covering a sufficient fraction of the provenance and at the same time provide sufficiently detailed information.

Efficient Summarization. While summarization of provenance has been studied in previous work, e.g., [2, 35], for why-not provenance we face the challenge that it is infeasible to generate full provenance as input for summarization. As mentioned above in our running example, there are

[2]We deliberately use the term placeholder and not variable to avoid confusion with the variables of a rule.

$\sim 15 \cdot 10^{12}$ derivations of missing answers matching Bob's question if we use F-Airbnb dataset. We overcome this problem by (i) integrating summarization with provenance capture and (ii) developing a method for sampling rule derivations from the why-not provenance without materializing it first. Our sampling technique is based on the observation that the number of missing answers is typically significantly larger than the number of existing answers. Thus, to create a sample of the why-not provenance of missing answers matching a provenance question, we can randomly generate derivations that match the provenance question. We, then, filter out derivations for existing answers. This approach is effective, because a randomly generated derivation is much more likely to derive a missing than an existing answer.

Contributions. To the best of our knowledge, we are the first to address both the usability and scalability (computational) challenges of why-not provenance through summaries. Specifically, we make the following contributions:

- Using patterns, we generate meaningful summaries for the why and why-not provenance of unions of conjunctive queries with negation and inequalities ($\mathsf{UCQ}^{\neg<}$).
- We develop a summarization algorithm that applies sampling during provenance capture and avoids enumerating full why-not provenance. Our approach outsources most computation to a database system.
- We experimentally compare our approach with a single-derivation approach and Artemis [13], demonstrating that it efficiently produces high-quality summaries.

The remainder of this paper is organized as follows. We cover preliminaries in Sec. 2 and define the provenance summarization problem in Sec. 3. We present an overview of our approach Sec. 4 and, then, discuss sampling, pattern candidate selection, and top-k summary construction (Sec. 5 to 8). We present experiments in Sec. 9, discuss related work in Sec. 10, and conclude in Sec. 11.

2. BACKGROUND

2.1 Datalog

A Datalog program Q consists of a finite set of rules $r : \mathtt{R}(\overline{X}) :- \mathtt{g_1}(\overline{X_1}), \ldots, \mathtt{g_m}(\overline{X_n})$ where $\overline{X_j}$ denotes a tuple of variables and/or constants. $\mathtt{R}(\overline{X})$ is the *head* of the rule, denoted as $\mathsf{head}(r)$, and $\mathtt{g_1}(\overline{X_1}), \ldots, \mathtt{g_m}(\overline{X_n})$ is the *body* (each $\mathtt{g_j}(\overline{X_j})$ is a *goal*). We use $\mathsf{vars}(r)$ to denote the set of variables in r. The set of relations in the schema over which Q is defined is referred to as the extensional database (EDB), while relations defined through rules in Q form the intensional database (IDB), i.e., the heads of rules. All rules r of Q have to be *safe*, i.e., every variable in r must occur in a positive literal in r's body. Here, we consider union of conjunctive queries with negation and comparison predicates or $\mathsf{UCQ}^{\neg<}$. Thus, all rules of a query have the same head predicate and goals in the body are either *literals*, i.e., atoms $\mathtt{L}(\overline{X_j})$ or their negation $\neg\mathtt{L}(\overline{X_j})$. The body may have comparisons of the form $a \diamond b$ where a and b are either constants or variables and $\diamond \in \{<, \leq, \neq, \geq, >\}$. For example, considering the Datalog rule r_1 in Fig. 1, $\mathsf{head}(r_1)$ is $\mathtt{AL}(N, R)$ and $\mathsf{vars}(r_1)$ is $\{I, N, T, R, E, P\}$. The rule is safe since all head variables occur in the body and all goals are positive.

The active domain $adom(D)$ of a database D (an instance of EDB relations) is the set of all constants that appear in D. We assume the existence of a universal domain of values $\mathbb{D}$ which is a superset of the active domain of every database. The result of evaluating Q over D, denoted as $Q(D)$, contains all IDB tuples $Q(t)$ for which there exists a successful rule derivation with head $Q(t)$. A derivation of r is the result of applying a valuation $\nu : \mathsf{vars}(r) \to \mathbb{D}$ which maps the variables of r to constants such that all comparisons of the rule hold, i.e., for each comparison $\psi(\overline{Y})$ the expression $\psi(\nu(\overline{Y}))$ evaluates to true. Note that the set of all derivations of r is independent of D since the constants of a derivation are from $\mathbb{D}$. Let $\overline{c}$ be a list of constants from $\mathbb{D}$, one for each variable of r. We use $r(\overline{c})$ to denote the rule derivation that assigns constant c_i to variable X_i in r. Note that variables are ordered by the position of their first occurrence in r, e.g., the variable order for r_1 (Fig. 1) is (N, R, I, T, E, P). A rule derivation is successful (failed) if all (at least one of) the goals in its body are successful (is failed). A positive/negative literal goal is successful if the corresponding tuple exists/doesn't exist. A missing answer for Q and D is an IDB tuple $Q(t)$ for which all derivations are failed. For a given D and r, we use $D \models r(\overline{c})$ to denote that $r(\overline{c})$ is successful over D. Typically, as mentioned in Sec. 1, not all failed derivations constructed in this way are sensible, e.g., a derivation may assign an integer to an attribute of type string. We allow users to control which values to consider for which attribute (see [20, 22]). For simplicity, however, we often assume a single universal domain $\mathbb{D}$.

2.2 Provenance Model

We now explain the provenance model introduced in Ex. 2. As demonstrated in [22], this provenance model is equivalent to the provenance semiring model for positive queries [11] and to its extension for first-order (FO) formula [27]. In our model, existing IDB tuples are connected to the successful rule derivations that derive them while missing tuples are connected to all failed derivations that could have derived them. Successful derivations are connected to successful goals. Failed derivations are only connected to failed goals (which justify the failure). Nodes in provenance graphs carry two types of labels: (i) a label that determines the node type (tuple, rule, or goal) and additional information, e.g., the arguments and rule identifier of a derivation, and (ii) a label indicating success/failure. We encode (ii) as colors in visualizations of such graphs. As shown in [20], provenance in this model can equivalently be represented as sets of successful and failed rule derivations as long as the success/failure state of goals are known.

Definition 1 (Annotated Rule Derivation). *Let D be a database and $r : \mathtt{Q}(\overline{X}) :- \mathtt{R_1}(\overline{X_1}), \ldots, \mathtt{R_1}(\overline{X_l}), \neg\, \mathtt{R_{l+1}}(\overline{X_{l+1}}), \ldots, \neg\, \mathtt{R_m}(\overline{X_n}), \psi(\overline{Y_1}), \ldots, \psi(\overline{Y_k})$ a Datalog rule where ψ_i is a comparison. An annotated derivation $d = r(\overline{c}) - (\overline{g})$ of r consists of a list of constants $\overline{c}$ and a list of goal annotations $\overline{g} = (g_1, \ldots, g_m)$ such that (i) $r(\overline{c})$ is a rule derivation, and (ii) $g_i = T$ if $i \leq l \wedge D \models R_i(\overline{c_i})$ or $i > l \wedge D \not\models R_i(\overline{c_i})$ and $g_i = F$ otherwise.*

An example failed annotated derivation of rule r_1 (Fig. 1) is $d_1 = r_1(\mathsf{cp\!: central\ place, shared, 8403, apt, east, 130}) - (T, F)$ from Fig. 2a. That is, while $\mathtt{A}(\mathsf{8403, 2016\text{-}11\text{-}09, 130})$ failed, $\mathtt{L}(\mathsf{8403, cp\!: central\ place, apt, shared, queen\ anne, east})$ is successful. Note that the constants 2016-11-09 and queen anne are part of rule r_1. Only assignments to variables are listed in rule derivations. Using annotated derivations, we can explain the existence or absence of a (set of) query result tuple(s). We use $\mathcal{A}(Q, D, r)$ to denote all annotated derivations of rule r from Q according to D, $\mathcal{A}(Q, D)$ to denote $\bigcup_{r \in Q} \mathcal{A}(Q, D, r)$, and $\mathcal{A}(Q, D, t)$ to denote the subset of $\mathcal{A}(Q, D)$ with head $Q(t)$. We would like to emphasize that valuations that violate any comparison of a rule are not considered to be rule derivations.

We now define *provenance questions (PQ)*. Through the type of PQ (WHY or WHYNOT), the user specifies whether she is interested in missing or existing results. In addition, the user provides a tuple $\boldsymbol{t}$ of constants (from $\mathbb{D}$) and placeholders to indicate what tuples she is interested in. We refer to such tuples as pattern tuples (*p-tuples* for short) and use bold font to distinguish them from tuples with constants only. We use capital letters to denote placeholders and variables, and lowercase to denote constants. We say a tuple t *matches* a p-tuple $\boldsymbol{t}$, written as $t \preccurlyeq \boldsymbol{t}$, if we can unify t with $\boldsymbol{t}$ by applying a valuation ν that substitutes placeholders in $\boldsymbol{t}$ with constants from $\mathbb{D}$ such that $\nu(\boldsymbol{t}) = t$, e.g., $\texttt{AL}(\mathsf{plum}, \mathsf{shared}) \preccurlyeq \texttt{AL}(N, \mathsf{shared})$ using $\nu := N \rightarrow \mathsf{plum}$. The provenance of all existing (missing) tuples matching $\boldsymbol{t}$ constitutes the answer of a WHY (WHYNOT) PQ.

DEFINITION 2 (PROVENANCE QUESTION). *Given a query Q, a provenance question Φ over Q is a pair $(\boldsymbol{t}, \textsf{type})$ where $\boldsymbol{t}$ is a p-tuple and $\textsf{type} \in \{\textsc{Why}, \textsc{Whynot}\}$.*

Bob's question from Ex. 1 can be written as $\Phi_{bob} = (\boldsymbol{t}_{bob}, \textsc{Whynot})$ where $\boldsymbol{t}_{bob} = \texttt{AL}(N, \mathsf{shared})$, i.e., Bob wants an explanation for *all* missing answers where $R = \mathsf{shared}$. The graph shown in Fig. 2a is part of the provenance for Φ_{bob}.

DEFINITION 3 (PROVENANCE). *Let Q be an n-nary $\mathsf{UCQ}^{\neg<}$ query, D a database, and $\boldsymbol{t}$ an n-nary p-tuple. We define the why and why-not provenance of $\boldsymbol{t}$ over Q and D as:*

$$\textsc{Why}(Q, D, \boldsymbol{t}) = \bigcup_{t \preccurlyeq \boldsymbol{t} \wedge t \in Q(D)} \textsc{Why}(Q, D, t)$$

$$\textsc{Why}(Q, D, t) = \{d \mid d \in \mathcal{A}(Q, D, t) \wedge D \models d\}$$

$$\textsc{Whynot}(Q, D, \boldsymbol{t}) = \bigcup_{t \preccurlyeq \boldsymbol{t} \wedge t \notin Q(D)} \textsc{Whynot}(Q, D, t)$$

$$\textsc{Whynot}(Q, D, t) = \{d \mid d \in \mathcal{A}(Q, D, t) \wedge D \not\models d\}$$

The provenance $\textsc{Prov}(\Phi)$ of a provenance question Φ is:

$$\textsc{Prov}(\Phi) = \begin{cases} \textsc{Why}(Q, D, \boldsymbol{t}) & \textit{if } \Phi = (\boldsymbol{t}, \textsc{Why}) \\ \textsc{Whynot}(Q, D, \boldsymbol{t}) & \textit{if } \Phi = (\boldsymbol{t}, \textsc{Whynot}) \end{cases}$$

3. PROBLEM DEFINITION

We now formally define the problem addressed in this work: how to summarize the provenance $\textsc{Prov}(\Phi)$ of a provenance question Φ. For that, we introduce derivation patterns that concisely describe provenance and, then, define provenance summaries as sets of such patterns. We also develop quality metrics for such summaries that model completeness and informativeness as introduced in Sec. 1.

3.1 Derivation pattern

A *derivation pattern* is an annotated rule derivation whose arguments can be both constants and placeholders.

DEFINITION 4 (DERIVATION PATTERN). *Let r be a rule with n variables and m goals and $\mathbb{P}$ an infinite set of placeholders. A* derivation pattern *$p = r(\bar{e}) - (\bar{g})$ consists of a list $\bar{e}$ of length n where $e_i \in \mathbb{D} \cup \mathbb{P}$ and $\bar{g}$, a list of m booleans.*

Consider pattern $p_1 = r_1(N, \mathsf{shared}, I, \mathsf{apt}, E, P) - (T, F)$ for rule r_1 (Fig. 1) shown in Fig. 2b. Pattern p_1 represents the set of failed derivations matching $\texttt{AL}(N, \mathsf{shared})$ where the listing is an apartment (apt) and for which the 1^{st} goal succeeded (the listing exists in Queen Anne) and the 2^{nd} goal failed (the listing is not available on Nov 9th, 2016). In the following, we will use $p[i]$ to denote the ith argument of pattern p. We omit the goal annotations of patterns if they are irrelevant to the discussion. We call p a pattern for a p-tuple $\boldsymbol{t}$ if p and $\boldsymbol{t}$ agree on constants, e.g., p_1 is a pattern for $\boldsymbol{t}_{bob} = \texttt{AL}(N, \mathsf{shared})$ since $p[2] = \boldsymbol{t}_{bob}[2] = \mathsf{shared}$. We use $\textsc{Pat}(Q, \boldsymbol{t})$ to denote the set of all patterns for $\boldsymbol{t}$ and Q.

3.2 Pattern Matches

A derivation pattern p represents the set of derivations that "match" the pattern. We define *pattern matches* as valuations that replace the placeholders in a pattern with constants from $\mathbb{D}$. In the following, we use $\mathsf{placeh}(p)$ to denote the set of placeholders of a pattern p.

DEFINITION 5 (PATTERN MATCHES). *A derivation pattern $p = r(\bar{e}) - (\bar{g_1})$ matches an annotated rule derivation $d = r(\bar{c}) - (\bar{g_2})$, written as $p \preccurlyeq d$, if there exists a valuation $\nu : \mathsf{placeh}(p) \rightarrow \mathbb{D}$ such that $\nu(p) = d$ and $\bar{g_1} = \bar{g_2}$.*

Consider $p_1 = r_1(N, \mathsf{shared}, I, \mathsf{apt}, E, P) - (T, F)$ and $d_1 = r_1(\mathsf{cp\colon\ central\ place}, \mathsf{shared}, 8403, \mathsf{apt}, \mathsf{east}, 130) - (T, F)$ (in Fig. 2a and 2b). We have $p_1 \preccurlyeq d_1$ since the valuation $N \rightarrow \mathsf{cp}$, $I \rightarrow 8403$, $E \rightarrow \mathsf{east}$, and $P \rightarrow 130$ maps p_1 to d_1 and the goal indicators (T, F) are same for p_1 and d_1.

3.3 Provenance Summary

We define provenance summaries to be sets of patterns.

DEFINITION 6 (PROVENANCE SUMMARY). *Let Q be a $\mathsf{UCQ}^{\neg<}$ query and $\Phi = (\boldsymbol{t}, \textsf{type})$ a provenance question. A* provenance summary *$\mathcal{S}$ for Φ is a subset of $\textsc{Pat}(Q, \boldsymbol{t})$.*

Based on the Def. 6, any subset of $\textsc{Pat}(Q, \boldsymbol{t})$ is a summary. However, summaries do differ in conciseness, informativeness, and completeness. Consider a summary for Φ_{bob} consisting of $p_2 = r_1(N, \mathsf{shared}, I, T, E, P) - (T, F)$ and $p_2' = (N, \mathsf{shared}, I, T, E, P) - (F, F)$. This summary fully covers $\textsc{Prov}(\Phi_{bob})$.[3] However, since the pattern only consists of placeholders and constants from Φ_{bob}, no new information is conveyed. An example of the other extreme is $p_3 = r_1(\mathsf{cp\colon\ central\ place}, \mathsf{shared}, 8403, \mathsf{apt}, \mathsf{east}, 130) - (T, F)$. The pattern consists only of constants and, thus, provides detailed information but covers only a single derivation.

3.4 Quality Metrics

We now introduce a quality metric that combines *completeness* and *informativeness*. We define *completeness* as the fraction of $\textsc{Prov}(\Phi)$ matched by a derivation pattern based on the Def. 5. For a question Φ, query Q, and database D, we use $\mathcal{M}(Q, D, p, \Phi)$ to denote all derivations in $\textsc{Prov}(\Phi)$ that match a pattern p:

$$\mathcal{M}(Q, D, p, \Phi) := \{d \mid d \in \textsc{Prov}(\Phi) \wedge d \preccurlyeq p\}$$

Considering the pattern p_1 from Fig. 2b and the derivation d_1 from Fig. 2a, we have $d_1 \in \mathcal{M}(r_1, D, p_1, \Phi_{bob})$.

[3] Pattern $p_2'' = r_1(N, \mathsf{shared}, I, T, E, P) - (F, T)$ has no matches, because non-existing listings cannot be available.

DEFINITION 7 (COMPLETENESS). *Let Q be a query, D a database, p a pattern, and Φ a provenance question. The* completeness *of p is defined as* $cp(p) = \frac{|\mathcal{M}(Q,D,p,\Phi)|}{|\textsc{Prov}(\Phi)|}$.

We also define *informativeness* which measures how much new information is conveyed by a pattern.

DEFINITION 8 (INFORMATIVENESS). *For a pattern p and question Φ with p-tuple $\boldsymbol{t}$, let $arity(p)$ denote the arity of p and $C(p)$ and $C(\boldsymbol{t})$ the number of constants in p and $\boldsymbol{t}$, respectively. The* informativeness *of p is* $info(p) = \frac{C(p)-C(\boldsymbol{t})}{arity(p)-C(\boldsymbol{t})}$.

For Bob's question Φ_{bob} and pattern $p_1 = r_1(N, \mathsf{shared}, I, \mathsf{apt}, E, P)$, we have $info(p_1) = 0.2$ because $C(p_1)$ is 2 (shared and apt), $C(\boldsymbol{t}_{bob})$ is 1 (shared), and $arity(p_1)$ is 6 (all placeholders and constants). We generalize completeness and informativeness to sets of patterns (summaries) as follows. The completeness of a summary $\mathcal{S}$ is the fraction of the $\textsc{Prov}(\Phi)$ covered by at least one pattern. For patterns p_2 and p'_2 from Sec. 3.3, we have $cp(\{p_2, p'_2\}) = cp(p_2) + cp(p'_2) = 1$. Note that $cp(\mathcal{S})$ may not be equal to the sum of $cp(p)$ for $p \in \mathcal{S}$, since match sets of patterns may overlap. We will revisit overlap in Sec. 8. We define informativeness as the average informativeness of the patterns in $\mathcal{S}$.

$$cp(\mathcal{S}) = \frac{|\bigcup_{p\in\mathcal{S}} \mathcal{M}(Q,D,p,\Phi)|}{|\textsc{Prov}(\Phi)|} \quad info(\mathcal{S}) = \frac{\sum_{p\in\mathcal{S}} info(p)}{|\mathcal{S}|}$$

We define the score of a summary $\mathcal{S}$ as the harmonic mean of completeness and informativeness, i.e., $sc(\mathcal{S}) = 2 \cdot \frac{cp(\mathcal{S})\cdot info(\mathcal{S})}{cp(\mathcal{S})+info(\mathcal{S})}$. We are now ready to define the *top-k provenance summarization problem* which, given a provenance question Φ, returns the top-k patterns for Φ wrt. $sc(\mathcal{S})$.

- **Input:** A query Q, database D, provenance question $\Phi = (\boldsymbol{t}, \mathsf{type})$, $k \in \mathbb{N} \geq 1$.
- **Output:** $\mathcal{S}(Q,D,\Phi,k) = \underset{\mathcal{S}\subset\textsc{Pat}(Q,\boldsymbol{t})\wedge|\mathcal{S}|=k}{\operatorname{argmax}} sc(\mathcal{S})$

4. OVERVIEW

Before describing our approach in detail in the following sections, we first give a brief overview of each step in the computation. To compute the top-k provenance summary $\mathcal{S}(Q,D,\Phi,k)$ for a provenance question Φ, we have to (i) compute provenance for Φ, (ii) enumerate all patterns that could be used in summaries, (iii) calculate matches between derivations and the patterns to calculate completeness of sets of patterns, and (iv) find the set of k patterns that has the highest score among all sets of patterns of size k for Φ. To compute the exact solution to this problem, we need to enumerate all derivations from the why or why-not provenance. However, as mentioned in the introduction, the why-not provenance can be very large. Specifically, as we will discuss further in the following section, it's size is in $O(|\mathbb{D}|^n)$, i.e., linear in the size of the data domain $\mathbb{D}$, but exponential in n, the maximal number of variables of a rule from Q that is not bound to constants by Φ. Thus, solving the problem exactly is infeasible for why-not provenance. Instead, we present a sampling-based approach that outsources most of the computation to a database for scalability.

Sampling Provenance. For why-not provenance, we apply the sampling technique described in Sec. 5 to compute

Query: $r_{ex} : \mathtt{Q_{ex}}(X,Y) :- \mathtt{R}(X,Z), \mathtt{R}(Z,Y), X < Y$

PQ: $\Phi_{ex} = (\boldsymbol{t}_{ex}, \textsc{Whynot})$ where $\boldsymbol{t}_{ex} = \mathtt{Q}(A, 4)$

Query Unified With P-Tuple $\boldsymbol{t}_{ex}$:

$$r_{ex}^{\Phi_{ex}} : \mathtt{Q_{ex}^{\Phi_{ex}}}(X,4) :- \mathtt{R}(X,Z), \mathtt{R}(Z,4), X < 4$$

R

A	B
1	2
2	3
2	4
5	3
5	5
5	6

$\mathbf{Q}_{ex}$

A	B
1	3
1	4
5	6

Answers matching $\boldsymbol{t}_{ex}$

A	B
1	4
2	4
3	4

Figure 3: Running Example for Summarization

an unbiased sample S of derivations from the why-not provenance. For why-provenance, we employ our query instrumentation technique from [22, 20] to produce a Datalog program whose output is the provenance graph for Φ.

Enumerating Pattern Candidates. The number of patterns for a rule of a Datalog program with m goals and n variables is in $O((|\mathbb{D}+n|^n \cdot 2^m)$. Even if we only consider patterns that match at least one derivation from the sample we produce, the number of patterns may still be a factor of 2^n larger than the sample. We adopt a heuristic from [9] that, in the worst case, generates quadratically many patterns (in the size of the sample S).

Estimating Pattern Coverage. To be able to compute the completeness metric for a set of patterns which is required for scoring sets of patterns in the last step, we need to determine what derivations are covered by which pattern and which of these derivations belong to the why-not provenance for Φ. Since it is only feasible to compute a sample S of the provenance, we estimate completeness based on S.

Computing the Top-k Summary. In the last step of the computation, we generate sets of patterns of size k from the set of patterns produced in the previous step, rank them based on their scores and return the set with the highest score as the top-k provenance summary. We apply a best-first search method and derive efficiently computable bounds for the completeness of sets of patterns to deal with a potentially large number of candidate summaries.

5. SAMPLING WHY-NOT PROVENANCE

In this section, we first discuss how to efficiently generate a sample of annotated derivations of a given size n_S from the why-not provenance for a provenance question (PQ). This sample will then be used in the following phases of our summarization algorithm. For instance, consider the example query r_{ex} shown on the top of Fig. 3 which returns endpoints of paths of length 2 in a graph with integer node labels such that the end point is labeled with a lareger number than the start point. Evaluating r_{ex} over the example instance $\mathtt{R}$ from the same figure yields three results: $\mathtt{Q_{ex}}(1,3)$, $\mathtt{Q_{ex}}(1,4)$, and $\mathtt{Q_{ex}}(5,6)$. In this example, we want to explain missing answers of the form $\mathtt{Q_{ex}}(X,4)$, i.e., answering the provenance question Φ_{ex} shown in Fig. 3. Recall that, the why-not provenance $\textsc{Whynot}(Q,D,\boldsymbol{t})$ of p-tuple $\boldsymbol{t}$ consists of all derivations of tuples $t \notin Q(D)$ where $t \preccurlyeq \boldsymbol{t}$. Assuming that $\mathbb{D} = \{1,2,3,4,5,6\}$, on the bottom right of Fig. 3 we show all missing and existing answers matching $\boldsymbol{t}_{ex}$ (missing answers are shown with red background).

5.1 Naive Unbiased Sampling

To generate all derivations for these missing answers, we can bind the variables of each rule r of query Q to the constants from $\boldsymbol{t}$ to ensure that only derivations of results which match the PQ's p-tuple $\boldsymbol{t}$ are generated. We refer to this process as unifying Q with $\boldsymbol{t}$. For our running example, this yields the rule $r_{ex}^{\Phi_{ex}}$ shown in Fig. 3. The naive way to create a sample of derivations from $\text{WHYNOT}(Q, D, \boldsymbol{t})$ using this rule is to repeatably sample a value from $\mathbb{D}$ for each variable, then check whether (i) the predicates of the rule are fulfilled and (ii) the resulting rule derivation computes a missing answer. For example, for $r_{ex}^{\Phi_{ex}}$, we may choose $X = 2$ and $Z = 2$ and get a derivation $d_1 = r_{ex}^{\Phi_{ex}}(2, 2)$. Derivation d_1 fulfills the predicate $X < 4$ and its head $\mathtt{Q_{ex}}(2, 4)$ is a missing answer. Thus, d_1 belongs to the why-not provenance of $\boldsymbol{t}_{ex}$. Then, to get an annotated rule derivation, we determine its goal annotations by checking whether the tuples corresponding to the grounded goals of the rule exists in the database instance. For this example, the first goal $\mathtt{R}(2, 2)$ fails, but the second goal $\mathtt{R}(2, 4)$ succeeds. Note that in this process there are two potential ways for why we may fail to produce a derivation of the why-not provenance: (i) a predicate of the rule may be violated by the bindings generated in this way (e.g., if we would have chosen $X = 5$, then $X < 4$ would not have held) and (ii) the derivation may derive an existing answer, e.g., if $X = 1$ and $Z = 3$, we get the failed derivation $r_{ex}(1, 3)$ of the existing answer $\mathtt{Q_{ex}}(1, 4)$.

Analysis of Naive Sampling. If we repeat the process described above until it has returned n_{S} why-not derivations, then this produces an unbiased sample of the why-not provenance for a p-tuple $\boldsymbol{t}$. Note that, technically there is no guarantee that the process will ever terminate since it may repeatedly produce derivations that do not fulfill a predicate or derive existing answers. Typically, the amount of missing answers is significantly larger than the number of answers, i.e., $|\text{WHYNOT}(Q, D, \boldsymbol{t})| \gg |\mathcal{A}(Q, D, \boldsymbol{t}) - \text{WHYNOT}(Q, D, \boldsymbol{t})|$. Thus, any randomly generated derivation is with high probability in $\text{WHYNOT}(Q, D, \boldsymbol{t})$. We will explain how to deal with derivations that fail to fulfill predicates in Sec. 5.2.

Batch Sampling. A major shortcoming of the naive sampling approach is that it requires us to evaluate queries to test for every produced derivation d whether it derives a missing answer ($\mathsf{head}(d) \notin Q(D)$) and to determine its goal annotations by checking for each grounded goal $R(\vec{c})$ or $\neg R(\vec{c})$ whether $R(\vec{c}) \in D$. It would be more efficient to model sampling as a single batch computation that we can outsource to a database system and that can be fused into a single query with the other phases of the summarization process to avoid unnecessary round-trips between our system and the database. However, for batch sampling, we have to choose upfront how many samples to create, but not all such samples will end up being why-not provenance or fulfill the rule's predicates. To ensure with high probability that the batch computation returns at least n_{S} derivations, we use a larger sample size $n_{\text{OS}} \geq n_{\text{S}}$ such that the probability that the resulting sample contains at least n_{S} derivations from $\text{WHYNOT}(Q, D, \boldsymbol{t})$ is higher than a configurable threshold $P_{success}$ (e.g., 99.9%). We refer to this part of the process as *over-sampling*. We discuss how to generate a query that computes a sample of size n_{OS} in Sec. 5.2 and, then, discuss how to determine n_{OS} in Sec. 5.3.

5.2 Batch Sampling Using Queries

For simplicity, we limit the discussion to queries with a single rule, e.g., the query $r_{ex}^{\Phi_{ex}}$ from Fig. 3. We discuss queries with multiple rules at the end of this section. The query we generate to produce a sample of size n_{OS} consists of three steps: generating derivations, filtering derivations of existing answers, determining goal annotations.

1. Generating Derivations. We first generate a query that creates a random sample OS of n_{OS} derivations (not annotated) for which there exists an annotated version in $\mathcal{A}(Q, D, \boldsymbol{t})$. Consider a single rule r with m goals, n variables, and h head variables: $r : \mathtt{R}(\overline{X}) \mathrel{:-} \mathtt{g_1}(\overline{X_1}), \ldots, \mathtt{g_m}(\overline{X_n}), \psi_1(\overline{Y_1}), \ldots, \psi_k(\overline{Y_l})$. Let R_i be the relation accessed by goal $\mathtt{g_i}$, i.e., $\mathtt{g_i}(\overline{X_i})$ is either $R_i(\overline{X_i})$ or $\neg R_i(\overline{X_i})$. Let k be the number of head variables bound by the p-tuple $\boldsymbol{t}$ from the why-not provenance question Φ for which we are sampling. We use $\overline{Z} = Z_1, \ldots, Z_u$ to denote $u = n - k$ variables of r that are not bound by $\boldsymbol{t}$. Recall that, to only consider derivations matching $\boldsymbol{t}$, we unify the rule with $\boldsymbol{t}$ by binding variables in the rule to the corresponding constants from $\boldsymbol{t}$. We use $r_{\boldsymbol{t}}$ to denote the resulting unified rule. Note that we will describe our summarization techniques using derivations and patterns for $r_{\boldsymbol{t}}$. Patterns for r can be trivially reconstructed from the results of summarization by plugging in constants from $\boldsymbol{t}$. To generate derivations for such a query, we sample n_{OS} values for each unbound variable independently with replacement, and, then, combine the resulting samples into a sample of $\mathcal{A}(Q, D, \boldsymbol{t})$ modulo goal annotations. Predicates comparing constants with variables, e.g., $X < 4$ in $r_{ex}^{\Phi_{ex}}$, are applied before sampling to remove values from the domain of a variable that cannot result in derivations fulfilling the predicates. Similar to [22, 20], we assume that the user specifies the domain $\mathbb{D}_A$ for each attribute A as a unary query that returns $\mathbb{D}_A$ (we provide reasonable defaults to avoid overloading the user). We extend relational algebra with two operators to be able to express sampling. Operator SAMPLE_n returns n samples which are chosen uniformly random with replacement from the input of the operator. We use $\#_A$ to denote an operator that creates an integer identifier for each input row that is stored in a column A appended to the schema of the operator's input. For each variable $X \in \overline{Z}$ with $\mathsf{attrs}(X) = \{A_1, \ldots, A_j\}$ ($\mathsf{attrs}(X)$ denotes the set of attributes that variable X is bound to by the rule containing X), we create a query Q_X that unions the domains of these attributes, then applies predicates that compare X with constants, and then samples n_{OS} values.

$$Q_X \coloneqq \#_{id}(\text{SAMPLE}_{n_{\text{OS}}}(\sigma_{\theta_X}(\rho_X((\mathbb{D}_{A_1} \cup \ldots \cup \mathbb{D}_{A_j})))))$$

Here, θ_X is a conjunction of all the predicates from $r_{\boldsymbol{t}}$ that compare X with a constant. The purpose of $\#$ is to allow us to use natural join to "zip" the samples for the individual variables into bindings for all variables of $r_{\boldsymbol{t}}$:

$$Q_{bind} \coloneqq \sigma_{\theta_{join}}(Q_{Z_1} \bowtie \ldots \bowtie Q_{Z_u})$$

Here, θ_{join} is a conjunction of all predicates from $r_{\boldsymbol{t}}$ that compare two variables. Note that the selectivity of θ_{join} has to be taken into account when computing n_{OS} (discussed in Sec. 5.3). Each tuple in the result of Q_{bind} encodes the bindings for one derivation d of a tuple $t \preccurlyeq \boldsymbol{t}$.

EXAMPLE 4. *Consider unified rule $r_{ex}^{\Phi_{ex}}$ from Fig. 3. Assume that $\mathbb{D}_A = \mathbb{D}_B = \Pi_A(R) \cup \Pi_B(R)$ and $n_{\text{OS}} = 3$. Variable X is bound to attribute A and Z is bound to both A and*

B. Thus, we generate the following queries:

$$\mathbb{D}_A = \mathbb{D}_B \coloneqq \Pi_A(R) \cup \Pi_B(R)$$
$$Q_X \coloneqq \#_{id}(\text{SAMPLE}_{n_{OS}}(\sigma_{X<4}(\rho_X(\mathbb{D}_A))))$$
$$Q_Z \coloneqq \#_{id}(\text{SAMPLE}_{n_{OS}}(\rho_Z(\mathbb{D}_A \cup \mathbb{D}_B)))$$
$$Q_{bind} \coloneqq Q_X \bowtie Q_Z$$

Evaluated over the example instance this query may return:

Q_X

id	X
1	1
2	2
3	2

Q_Z

id	Z
1	4
2	2
3	4

Q_{bind}

id	X	Z
1	1	4
2	2	2
3	2	4

2. Filtering Derivations of Existing Answers. We now construct a query Q_{der}, which checks for each derivation $d \in \text{OS}$ for a tuple $t \preccurlyeq \boldsymbol{t}$ whether $t \notin Q(D)$ and only retain derivations passing this check. This is achieved by anti-joining Q_{bind} with Q which we restricted to tuples matching $\boldsymbol{t}$ since only such tuples can be derived by Q_{bind}. We construct a query Q_{der} for this step:

$$Q_{der} \coloneqq Q_{bind} \triangleright_{\theta_{der}} \sigma_{\theta_{\boldsymbol{t}}}(Q)$$

The query Q_{der} uses condition θ_{der} which equates attributes from Q_{bind} that correspond to head variables of $r_{\boldsymbol{t}}$ with the corresponding attribute from Q and condition $\theta_{\boldsymbol{t}}$ that filters out derivations not matching $\boldsymbol{t}$ by equating attributes with constants from $\boldsymbol{t}$. For instance, for our running example, we remove answers from Q_{ex} where $Y \neq 4$ (since $\boldsymbol{t}_{ex}$ binds $Y = 4$) and anti-join on X, the only head variable of $r_{ex}^{\Phi_{ex}}$. The resulting query and result are shown below. Note that tuple $(1, 1, 4)$ was removed since it corresponds to a (failed) derivation of the existing answer $\mathtt{Q_{ex}}(1, 4)$.

$$Q_{der} \coloneqq Q_{bind} \triangleright_{X=X} \sigma_{Y=4}(Q_{ex})$$

id	X	Z
2	2	2
3	2	4

3. Computing Goal Annotations. Next, we determine goal annotations for each derivation to create a set of annotated derivations from $\text{WHYNOT}(Q, D, \boldsymbol{t})$. Recall that a positive (negative) grounded goal is successful if the corresponding tuple exists (is missing). We can check this by outer-joining the derivations with the relations from the rule's body. Based on the existence of a join partner, we create boolean attributes storing g_i for $1 \leq i \leq |\mathsf{body}(r)|$ (F is encoded as false). For a negated goal, we negate the result of the conditional expression such that F is used if a join partner exists. We construct query Q_{sample} shown below to associates derivations in Q_{der} with the goal annotations $\bar{g}$.

$$Q_{sample} \coloneqq \delta(\Pi_{Z_1,\cdots,Z_u,e_1\rightarrow g_1,\cdots,e_n\rightarrow g_m}(Q_{goals}))$$
$$Q_{goals} \coloneqq Q_{der} ⟕_{\theta_1} \Pi_{\mathbf{R_1},1\rightarrow h_1}(\mathtt{R_1}) \ldots ⟕_{\theta_n} \Pi_{\mathbf{R_1},1\rightarrow h_m}(\mathtt{R_m})$$

Note that we use duplicate elimination to preserve set semantics. In projection expressions, we use $e \rightarrow a$ to denote projection on a scalar expression e whose result is stored in attribute a. Here, the join condition θ_i equates the attributes storing the values from $\overline{X_i}$ in $r_{\boldsymbol{t}}$ with the corresponding attributes from R_i. Attributes at positions that are bound to constants in $r_{\boldsymbol{t}}$ are equated with the constant. The net effect is that a tuple from Q_{der} corresponding to a rule derivation d has a join partner in R_i iff the tuple corresponding to the i^{th} goal of d exists in D. The expression e_i used in the projection of Q_{sample} then computes the boolean indicator for goal g_i as follows:

$$e_i \coloneqq \begin{cases} \textbf{if } (\textbf{isnull}(h_i)) \textbf{ then } T \textbf{ else } F & \text{if } g_i \text{ is positive} \\ \textbf{if } (\textbf{isnull}(h_i)) \textbf{ then } F \textbf{ else } T & \text{otherwise} \end{cases}$$

EXAMPLE 5. *For our running example, we generate:*

$$Q_{sample} \coloneqq \delta(\Pi_{X,Z,\textbf{if } (\textbf{isnull}(h_1)) \textbf{ then } T \textbf{ else } F\rightarrow g_1, \textbf{if } (\textbf{isnull}(h_2)) \textbf{ then } T \textbf{ else } F\rightarrow g_2}(Q_{goals}))$$
$$Q_{goals} \coloneqq Q_{der} ⟕_{X=A\wedge Z=B} \Pi_{A,B,1\rightarrow h_1}(R)$$
$$⟕_{Z=A\wedge B=4} \Pi_{A,B,\rightarrow h_2}(R)$$

Evaluating this query, we get the result shown below.

Q_{sample}

id	X	Z	h_1	h_2
2	2	2	F	T
3	2	4	T	F

The first tuple corresponds to the derivation $r_{ex}^{\Phi_{ex}}(2, 2) - (F, T)$ for which the first goal fails since $\mathtt{R}(2, 2)$ does not exist in $\mathtt{R}$ while the second goal succeeds because $\mathtt{R}(2, 4)$ exists. Similarly, the second tuple corresponds to $r_{ex}^{\Phi_{ex}}(2, 4) - (T, F)$ for which the first goal succeeds since $\mathtt{R}(2, 4)$ exists while the second goal fails because $\mathtt{R}(4, 4)$ does not exist.

Queries With Multiple Rules. For queries with multiple rules, we determine n_{OS} separately for each rule (recall that we consider $\mathsf{UCQ}^{\neg<}$ queries where every rule has the same head predicate) and create a separate sample query for each rule as described above and also generate patterns separately for each rule. In the final step, we then select the top-k summary from the union of all these patterns.

Complexity. The runtime of our algorithm is linear in n_{OS} and $|D|$ which significantly improves over the naive algorithm which is in $O(|\mathbb{D}|^n)$.

Implementation. Some DBMS such as Oracle and Postgres support a sample operator out of the box which we can use to implement the SAMPLE operator introduced above. However, these implementations of a sampling operator do not support sampling with replacement out of the box. We can achieve decent performance for sampling with replacement using a set-returning function that takes as input the result of applying the built-in sampling operator to generate a sample of size n_{OS}, caches this sample, and then samples n_{OS} times from the cached sample with replacement. The $\#_A$ operator can be implemented in SQL using `ROW_NUMBER()`. The expressions **if** (θ) **then** e_1 **else** e_2 and **isnull**() can be expressed in SQL using `CASE WHEN` and `IS NULL`, respectively.

5.3 Determining Over-sampling Size

We now discuss how to choose n_{OS}, the size of the sample OS produced by query Q_{bind}, such that the probability that OS contains at least n_S derivations from $\text{WHYNOT}(Q, D, \boldsymbol{t})$ is higher than a threshold $P_{success}$ under the assumption that the sampling method we introduced above samples uniformly random from $\mathcal{A}(Q, D, \boldsymbol{t})$. We then prove that our sampling method returns a uniform random sample. First, consider the probability p_{prov} that a uniform randomly chosen derivation from $\mathcal{A}(Q, D, \boldsymbol{t})$ is in $\text{WHYNOT}(Q, D, \boldsymbol{t})$ which is equal to the faction of derivations from $\mathcal{A}(Q, D, \boldsymbol{t})$ that is in $\text{WHYNOT}(Q, D, \boldsymbol{t})$:

$$p_{prov} = \frac{|\text{WHYNOT}(Q, D, \boldsymbol{t})|}{|\mathcal{A}(Q, D, \boldsymbol{t})|}$$

$|\mathcal{A}(Q, D, \boldsymbol{t})|$ can be computed from Q, $\boldsymbol{t}$, and the attribute domains as explained in Sec. 2.2. For instance, consider $r_{ex}^{\Phi_{ex}}$ without the conditional predicate in our running example and domain $\mathbb{D} = \{1, 2, 3, 4, 5, 6\}$. Then, there are $|\mathbb{D}|^n = 6^2$ possible derivations because 2 variables X and Z are not bound by $\boldsymbol{t}_{ex}$ and $\mathbb{D}$ has 6 values. To determine $|\text{WHYNOT}(Q, D, \boldsymbol{t})|$, we need to know how many derivations in $\mathcal{A}(Q, D, \boldsymbol{t})$ correspond to missing tuples matching $\boldsymbol{t}$. Since in most cases the number of missing answers vastly outweighs the number of existing tuples, it is more effective to compute the number of (successful and/or failed) derivations of $t \in Q(D)$ with $t \preccurlyeq \boldsymbol{t}$, i.e., $|\{t \mid t \in Q(D) \wedge t \preccurlyeq \boldsymbol{t}\}|$. This gives us the probability $p_{notProv}$ that a derivation is not in $\text{WHYNOT}(Q, D, \boldsymbol{t})$ and we get: $p_{prov} = 1 - p_{notProv}$.

Next, consider a random variable X that is the number of derivations from $\text{WHYNOT}(Q, D, \boldsymbol{t})$ in OS. We want to compute the probability $p(X \geq n_\text{S})$. For that, consider first $p(X = i)$, the probability that the sample OS we produce contains exactly i derivations from $\text{WHYNOT}(Q, D, \boldsymbol{t})$. We can apply standard results from statistics for computing $p(X = i)$, i.e., out of a sequence of n_OS picks with probability p_{prov} we get more than n_S successes. The probability to get exactly k successes out of n picks is $\binom{n}{k} \cdot {p_{prov}}^k \cdot (1 - p_{prov})^{n-k}$ based on the *Binomial Distribution*. For $i \neq j$, the events $X = i$ and $X = j$ are disjoint (it is impossible to have both exactly i and j derivations from $\text{WHYNOT}(Q, D, \boldsymbol{t})$ in OS). Thus, $p(X \geq n_\text{S})$ is $\sum p(X = i)$ for $i \in \{n_\text{S}, \ldots, n_\text{OS}\}$:

$$p(X \geq n_\text{S}) = \sum_{i=n_\text{S}}^{n_\text{OS}} \binom{n_\text{OS}}{i} \cdot {p_{prov}}^i \cdot (1 - p_{prov})^{n_\text{OS}-i}$$

Given p_{prov}, n_S, and $P_{success}$, we can compute the sample size n_OS such that $p(X \geq n_\text{S})$ is larger than $P_{success}$ ([1, 30] presents an algorithm for finding the minimum such n_OS).

Handling Predicates. Recall that we apply predicates that compare a variable with a constant before creating a sample for a variable. Thus, we do not need to consider these predicates when determining n_OS. Predicates of the form $X \diamond Y$ are applied after creating derivations. We estimate the selectivity of such predicates using standard techniques to estimate how many derivations will be filtered out and, then, increase n_OS to compensate for this. For instance, for a predicate with 0.5 selectivity we would double n_OS.

5.4 Analysis of Sampling Bias

We now formally analyze if our approach creates a uniform sample of $\text{WHYNOT}(Q, D, \boldsymbol{t})$. We demonstrate this by analyzing the probability $p(d \in \text{S})$ for an arbitrary derivation $d \in \text{WHYNOT}(Q, D, \boldsymbol{t})$ to be in the sample S. If our approach is unbiased, then this probability should be independent of which d is chosen and for $d' \neq d$ the events $d \in \text{S}$ and $d' \in \text{S}$ should be independent.

THEOREM 1. *Given derivations $d, d' \in \text{WHYNOT}(Q, D, \boldsymbol{t})$ and sample sizes n_S and n_OS, $p(d \in \text{S}) = c$ where c is a constant that is independent of the choice of d. Furthermore, the events $d \in \text{S}$ and $d' \in \text{S}$ are independent of each other.*

PROOF. To prove the theorem, we have to demonstrate that none of the phases of sampling introduces bias. In the following, let $\mathcal{A} := \mathcal{A}(Q, D, \boldsymbol{t})$, $n_A := |\mathcal{A}|$, and $n_P := |\mathcal{P}|$ where $\mathcal{P} := \text{WHYNOT}(Q, D, \boldsymbol{t})$. Recall that the first phase of sampling generates a sample OS of size n_OS by independently creating samples for each unbound variable which are combined into a sample of $\mathcal{A}$. Consider first the case where $n_\text{OS} = 1$, i.e., we pick a single value from each domain. Let $\mathbb{D}_i$ denote the domain for unbound variable Z_i in the single rule r of query Q. Since we sample uniform from $\mathbb{D}_i$, each of the $|\mathbb{D}_i|$ values has a probability of $\frac{1}{|\mathbb{D}_i|}$ to be chosen. Since the sample for Z_i is chosen independently from Z_j for $i \neq j$, any particular derivation $d \in \mathcal{A}$ to be in OS is $p(d \in \text{OS}) = \frac{1}{|\mathbb{D}_1| \times \ldots \times |\mathbb{D}_u|} = \frac{1}{|\mathcal{A}|}$. For $n_\text{OS} > 1$, observe that each value in the sample of $\mathbb{D}_i$ is chosen independently. Thus, $p(d \in \text{OS}) = 1 - p(d \notin \text{OS}) = 1 - (1 - \frac{n_A - 1}{n_A})^{n_\text{OS}}$ (the last equivalence is based on $p(A \cap B) = p(A) \cdot p(B)$ when A and B are independent). Furthermore, this implies that $d \in \text{OS}$ is independent of $d' \in \text{OS}$ for $d \neq d'$. So far, we have established that $p(d \in \text{OS})$ is constant and the events for picking particular derivations are mutually independent. It remains to be shown that the same holds for a derivation $d \in \mathcal{P}$ and the sample S we derive from OS. Since OS is sampled from $\mathcal{A}$, it may contain derivations $d' \notin \mathcal{P}$. Our sampling algorithms filters such derivations. Let S_{np} denote the set of all such derivations from OS and $n_{np} = |\text{S}_{np}|$. Observe that for $i \neq j$ the events $n_{np} = i$ and $n_{np} = j$ are obviously disjoint since OS contains a fixed number of derivations not in $\mathcal{P}$. Furthermore, $\sum_{i=0}^{n_\text{OS}} p(n_{np} = i) = 1$ since OS has to contain anywhere from zero to n_OS such derivations. Thus, we can compute the probability $p(d \in \text{S})$ as the sum over $i = \{0, \ldots, n_\text{OS}\}$ of the probability that d is selected to be in S conditioned on the probability that $d \in \text{OS}$ (otherwise d cannot be in S) and that $n_{np} = i$.

$$p(d \in \text{S}) = \sum_{i=0}^{n_\text{OS}} p(d \in \text{S} \mid d \in \text{OS} \cap n_{np} = i) \cdot p(d \in \text{OS} \cap n_{np} = i)$$

Now, consider the individual probabilities in this formula. Let $p_1 = p(d \in \text{S} \mid d \in \text{OS} \cap n_{np} = i)$. If d is in OS and $n_{np} = i$, then there are $n_\text{OS} - i - 1$ derivations from $\mathcal{P}$ in OS. Our sampling algorithm selects uniformly n_S derivation in $\mathcal{P}$ from $\text{OS} - \text{S}_{np}$ if $n_\text{OS} - i > n_\text{S}$. Thus, the probability for our particular derivation d to be in the final result is:

$$p_1 = \frac{\min(n_\text{S}, n_\text{OS} - i)}{n_\text{OS} - i}$$

Next, consider $p_2 = p(d \in \text{OS} \cap n_{np} = i)$. Based on our observation above, any subset of derivations of $\mathcal{A}$ has the same probability to be returned as OS. Thus, p_2 can be computed as the fraction of subsets of $\mathcal{A}$ of size n_OS that contain i successful derivation(s), and d and $n_\text{OS} - i - 1$ other derivations from $\mathcal{P}$. Putting differently, how many of the ${n_A}^{n_\text{OS}}$ possible samples that could be produced by our algorithm contain d and exactly i derivations from S_{np}:

$$p_2 = \frac{{n_{np}}^i \cdot {n_P}^{n_\text{OS}-i-1}}{{n_A}^{n_\text{OS}}}$$

Observe, that the formulas for p_1 and p_2 only refer to constants that are independent of the choice of d. Thus, $p(d \in \text{S})$ is independent of the choice of d. □

6. GENERATING PATTERN CANDIDATES

We now explain the candidate generation step of our summarization approach. Consider a PQ $\Phi = (\boldsymbol{t}, \text{WHYNOT})$ for a query Q. For any rule r of Q, let n be the number of unbound variables, i.e., $|\mathsf{vars}(r_{\boldsymbol{t}})|$ where $r_{\boldsymbol{t}}$ is the unified rule for r and $\boldsymbol{t}$, and m be the number of goals in r. The number of possible patterns for $r_{\boldsymbol{t}}$ is in $O((|\mathbb{D}|+n)^n \cdot 2^m)$, because for each variable of $r_{\boldsymbol{t}}$ we can choose either a placeholder or a value from $\mathbb{D}$ and for each goal we have to pick one of two possible annotations (F or T). Note that the names of placeholders are irrelevant to the semantics of a pattern, e.g., patterns $p = (A, 3)$ and $p' = (B, 3)$ are equivalent (matching the same derivations). That is, we only have to decide which arguments of a pattern are placeholders and which arguments share the same placeholder. Thus, it is sufficient to only consider n distinct placeholders P_i when creating patterns for $r_{\boldsymbol{t}}$ with n variables.

Example 6. *Consider rule $r_{ex}^{\Phi_{ex}}$ from Fig. 3. Let $\mathbb{D} = \{1, 2, 3, 4, 5, 6\}$ and $\mathbb{P} = \{P_1, P_2\}$. Let us for now ignore goal annotations. Note that taking the predicate $X < 4$ into account, any pattern where $X \geq 4$ cannot possibly match any derivations for this rules and, thus, we only have to consider patterns where X is bound to a constant less than 4 or a placeholder. The set of viable patterns is:*

$$r_{ex}^{\Phi_{ex}}(P_1, P_2), r_{ex}^{\Phi_{ex}}(P_1, 1), \ldots, r_{ex}^{\Phi_{ex}}(P_1, 6), r_{ex}^{\Phi_{ex}}(1, P_2), \ldots, r_{ex}^{\Phi_{ex}}(6, P_2),$$
$$r_{ex}^{\Phi_{ex}}(2, 1), \ldots, r_{ex}^{\Phi_{ex}}(2, 6), \ldots r_{ex}^{\Phi_{ex}}(3, 1), \ldots, r_{ex}^{\Phi_{ex}}(3, 6)$$

The set contains 31 elements. Considering goal annotations (F, F), (F, T), and (T, F), we get $31 \cdot 3 = 93$ patterns.

Given the $O((|\mathbb{D}|+n)^n \cdot 2^m)$ complexity, it is not feasible to enumerate all possible patterns. Instead, we adapt the *Lowest Common Ancestor* (LCA) method [9, 10] for our purpose which generates a number of pattern candidates from the sample derivations produced in the previous step that is at most quadratic in n_{S}. Thus, this approach sacrifices completeness to achieve better performance. Given a set of derivations (tuples in the work from [9, 10]), the LCA method computes the cross-product of this set with itself and generates candidate explanations by generalizing each such pair. The rationale is that each pattern generated in this fashion will at least match two derivations (or one derivation for the special case where a derivation is paired with itself). In our adaptation, we match derivations on the goal annotations so that only derivation with the same success/failure status of goals are paired. For each pair of derivations $d_1 = (a_1, \ldots, a_n) - (\bar{g})$ and $d_2 = (b_1, \ldots, b_n) - (\bar{g})$, we generate a pattern $p = (c_1, \ldots, c_n) - (\bar{g})$. Each element c_i in p is determined by: if $a_i = b_i$ then $c_i = a_i$. That is, constants on which d_1 and d_2 agree in the same position are retained. Otherwise, c_i is a fresh placeholder.

Example 7. *Reconsider the unified rule $r_{ex}^{\Phi_{ex}}$ and instance* R *from Fig. 3. Two example annotated rule derivations are $d_1 = r_{ex}^{\Phi_{ex}}(2, 1) - (F, F)$ and $d_2 = r_{ex}^{\Phi_{ex}}(2, 2) - (F, F)$. LCA generalizes d_1 and d_2 to generate a pattern $p = r_{ex}^{\Phi_{ex}}(2, Z) - (F, F)$ because $d_1[1] = d_2[1] = 2$ (and, thus, this constants is retained) and $p[2] = Z$ since $d_1[2] = 1 \neq 2 = d_2[2]$.*

We apply LCA to the sample S created using Q_{sample} from Sec. 5.2. Using LCA, we avoid generating exponentially many patterns improving the runtime of pattern generation from $O(|\mathbb{D}|^n)$ to $O({n_{\mathrm{S}}}^2)$ where typically $n_{\mathrm{S}} \ll |\mathbb{D}|$. Furthermore, this optimization reduces the input size for the final stages of the summarization process leading to additional performance improvements. We demonstrate experimentally in Sec. 9 that LCA performs well in practice.

Implementation. We implement the LCA method as a query Q_{lca} joining the query Q_{sample} (the query producing S) with itself on a condition $\theta_{lca} \coloneqq \bigwedge_{i=0}^{m} g_i = g_i$ where m is the number of goals of the rule r of Q (recall that we create patterns for each rule of a query independent and merge in the final step). Patterns are generated using a projection on an expression A_{lca}, where the i^{th} argument of a pattern is determined as **if** $(X_i = X_i)$ **then** X_i **else** $NULL$. Note that the LCA method never generates patterns where the same placeholder appears more than once. Thus, it is sufficient to encode placeholders as `NULL` values.

$$Q_{lca} \coloneqq \delta(\Pi_{A_{lca}}(Q_{sample} \bowtie_{\theta_{lca}} Q_{sample}))$$

The query generated for our running example is:

$$Q_{lca} \coloneqq \delta(\Pi_{e_x \to X, e_Z \to Z}(Q_{sample} \bowtie_{(g_1=g_1) \wedge (g_2=g_2)} Q_{sample})$$
$$e_x \coloneqq \textbf{if } (X = X) \textbf{ then } X \textbf{ else } NULL$$
$$e_Z \coloneqq \textbf{if } (Z = Z) \textbf{ then } Z \textbf{ else } NULL$$

7. ESTIMATING COMPLETENESS

To generate a top-k summary in the next step, we need to calculate the informativeness (Def. 8) and completeness (Def. 7) quality metrics for sets of patterns. Informativeness can be computed from the pattern without accessing the data. Recall that completeness is computed as the fraction of provenance matched by a pattern: $cp(p) = \frac{|\mathcal{M}(Q,D,p,\Phi)|}{|\textsc{Prov}(\Phi)|}$. Since we can materialize neither $|\mathcal{M}(Q, D, p, \Phi)|$ nor $|\textsc{Prov}(\Phi)|$, we have to estimate their sizes. In this section, we focus on how to estimate the completeness of individual patterns. How to compute the completeness metric for sets of patterns will be discussed in Sec. 8.

To determine whether a derivation $d \in \textsc{Prov}(\Phi)$ with goal annotations $\bar{g_1}$ matches a pattern p with goal annotation $\bar{g_2}$ that is in $\mathcal{M}(Q, D, p, \Phi)$, we have to check that $\bar{g_1} = \bar{g_2}$ and a valuation exists that maps p to d. Then, we count the number of such derivations exists to compute $|\mathcal{M}(Q, D, p, \Phi)|$. The existence of a valuation can be checked in linear time in the number of arguments of p by fixing a placeholder order and, then, assigning to each placeholder in p the corresponding constant in d if a unique such constant exists. The valuation fails if p and d end up having two different constants at the same position.

Example 8. *Continuing with Ex. 7, we compute completeness of the pattern $p = r_{ex}^{\Phi_{ex}}(2, Z, 4) - (F, F)$. For sake of the example, assume that $\textsc{Prov}(\Phi_{ex})$ is:*

$$d_1 = r_{ex}^{\Phi_{ex}}(2, 1) - (F, F) \qquad d_2 = r_{ex}^{\Phi_{ex}}(2, 2) - (F, F)$$
$$d_3 = r_{ex}^{\Phi_{ex}}(2, 3) - (T, F) \qquad d_4 = r_{ex}^{\Phi_{ex}}(2, 4) - (T, F)$$
$$d_5 = r_{ex}^{\Phi_{ex}}(2, 5) - (F, F) \qquad d_6 = r_{ex}^{\Phi_{ex}}(2, 6) - (F, F)$$

The completeness of p is $cp(p) = \frac{2}{3}$ because p matches all 4 derivations (d_1, d_2, d_5, and d_6) for which both goals have failed by assigning Z to 1, 2, 5, or 6.

To estimate the completeness of a pattern p, we compute the number of matches of p with derivations from the sample S produced by Q_{sample} as discussed in Sec. 5. As long as S is an unbiased sample of $\textsc{Prov}(\Phi)$, then the fraction of derivations from S matching the pattern is an unbiased estimate of the completeness of the pattern. With Ex. 8, assume that we create a sample $\mathrm{S} = \{d_1, d_3, d_5\}$. Estimating the completeness of pattern p based on S we get $cp(p) \simeq \frac{2}{3}$.

Implementation. We generate a query Q_{match} which joins the query Q_{lca} generating pattern candidates with Q_{sample}, the query generating the sample derivations. Let r_t be the rule for which we are generating patterns and A be the result attributes of Q_{lca}. We count the number of matches per pattern by grouping on A:

$$Q_{match} \coloneqq \gamma_{A,count(*)}(Q_{lca} \bowtie_{\theta_{match}} Q_{sample})$$

Recall that we encode placeholders as `NULL` values. Condition θ_{match} is a conjunction of conditions, one for each argument X of the pattern/derivation: $X = X \vee \textbf{isnull}(X)$. Since the number of candidates produced by LCA is at most ${n_S}^2$, matching is in $O({n_S}^2 \cdot n_S) = O({n_S}^3)$. For our running example, we would create the following query:

$$Q_{match} \coloneqq \gamma_{X,Z,g_1,g_2,count(*)}(Q_{lca} \bowtie_{\theta_{match}} Q_{sample})$$
$$\theta_{match} \coloneqq (X = X \vee \textbf{isnull}(X)) \wedge (Z = Z \vee \textbf{isnull}(Z))$$

8. COMPUTING TOP-K SUMMARIES

We now explain how to compute a top-k provenance summary for a provenance question Φ. This is the only step that is evaluated on the client-side. It's input is the set of patterns (denoted as $\textsc{Pat}_{lca}$) with completeness estimates returned by evaluating query Q_{match} (Sec. 7). We have to find the set $\mathcal{S} \subseteq \textsc{Pat}_{lca}$ of size k that maximizes $sc(\mathcal{S})$. A brute force solution would enumerate all such subsets, compute their scores (which requires us to compute the union of the matches for each pattern in the set to compute completeness), and return the one with the highest score. However, the number of candidates is $\binom{|\textsc{Pat}_{lca}|}{k}$ and this would require us to evaluate a query to compute matches for each candidate. Our solution uses lower and upper bounds on the completeness of patterns that can be computed based on the patterns and their completeness alone to avoid running additional queries. Furthermore, we use a best-first search method to incrementally build candidate sets guiding the search using these bounds.

8.1 Pattern Generalization and Disjointness

In general, the exact completeness of a set of patterns cannot be directly computed based on the completeness of the patterns of the set, because the sets of derivations matching two patterns may overlap. We present two conditions that allow us to determine in some cases whether the match sets of two patterns are disjoint or one is contained in the other. We say a pattern p_2 *generalizes* a pattern p_1 written as $p_1 \preceq_p p_2$ if $\forall i : p_1[i] = p_2[i] \vee p_2[i] \in \mathbb{P}$, and they have the same goal annotations. For instance, $(X, Y, \mathsf{a}) - (F, F)$ generalizes $(X, \mathsf{b}, \mathsf{a}) - (F, F)$. From Sec. 3, it immediately follows that if $p_1 \preceq_p p_2$ then $\mathcal{M}(Q, D, p_1, \Phi) \subseteq \mathcal{M}(Q, D, p_2, \Phi)$ since any derivation matching p_1 also matches p_2 and, thus, $cp(\{p_1, p_2\}) = cp(p_2)$. We say pattern p_1 and p_2 are *disjoint* written as $p_1 \perp_p p_2$ if (i) they are from different rules, (ii) they do not share the same goal annotations, or (iii) there exists an i such that $p_1[i] = c_1 \neq c_2 = p_2[i]$. That is, the patterns have a different constant at the same position i. If $p_1 \perp_p p_2$, then $\mathcal{M}(Q, D, p_1, \Phi) \cap \mathcal{M}(Q, D, p_2, \Phi) = \emptyset$ and, thus, we have $cp(\{p_1, p_2\}) = cp(p_1) + cp(p_2)$. Note that for any $\mathcal{S}$, $cp(\mathcal{S})$ is trivially bounded from below by $\max_{p \in \mathcal{S}} cp(p)$ (making the worst-case assumption that all patterns fully overlap) and by $\min(1, \sum_{p \in \mathcal{S}} cp(p))$ from above (completeness is maximized if there is no overlap). Using generalization and disjointness, we can refine these bounds. Note that generalization is transitive. To use generalization to find tighter upper bounds on completeness for a pattern set $\mathcal{S}$, we compute the set $\mathcal{S}_{ub} = \{p \mid p \in \mathcal{S} \wedge \neg\exists p' \in \mathcal{S} : p \preceq_p p'\}$. Any pattern not in $\mathcal{S}_{ub}$ is generalized by at least one pattern from $\mathcal{S}_{ub}$. For disjointness, if we have a set of patterns $\mathcal{S}$ for which patterns are pairwise disjoint, then $cp(\mathcal{S}) = \sum_{p \in \mathcal{S}} cp(p)$. Based on this observation, we find the subset $\mathcal{S}_{lb}$ of pairwise disjoint patterns from $\mathcal{S}$ that maximizes completeness, i.e., $\mathcal{S}_{lb} = \text{argmax}_{\mathcal{S}' \subseteq \mathcal{S} \wedge \forall p \neq p' \in \mathcal{S}' : p \perp_p p'} \sum_{p \in \mathcal{S}'} cp(p)$.[4] We use $\mathcal{S}_{lb}$ and $\mathcal{S}_{ub}$ to define an lower bound $\underline{cp}(\mathcal{S})$ and upper-bound $\overline{cp}(\mathcal{S})$ on the completeness of a pattern set $\mathcal{S}$:

$$\underline{cp}(\mathcal{S}) \coloneqq \sum_{p \in \mathcal{S}_{lb}} cp(p) \qquad \overline{cp}(\mathcal{S}) \coloneqq \sum_{p \in \mathcal{S}_{ub}} cp(p)$$

EXAMPLE 9. *Consider the following patterns for $r_{ex}^{\Phi_{ex}}$ from Fig. 3: $p = (2, Z) - (F, F)$, $p' = (3, Z) - (F, F)$, $p'' = (2, 1) - (F, F)$. Assume that $cp(p) = 0.44$, $cp(p') = 0.55$, and $cp(p'') = 0.1$. Consider $\mathcal{S} = \{p, p', p''\}$ and observe that $p \perp_p p'$, $p' \perp_p p''$, and $p'' \preceq_p p$. Thus, $\mathcal{S}_{ub} = \{p, p'\}$ (the pattern p'' is generalized by p) and $\mathcal{S}_{lb} = \{p, p'\}$ (while also $p' \perp_p p''$ holds, we have $cp(p) + cp(p') > cp(p') + cp(p'')$). We get: $\underline{cp}(\mathcal{S}) = cp(p) + cp(p') = 0.99$ and $\overline{cp}(\mathcal{S}) = cp(p) + cp(p') = 0.99$ from which follows that $cp(\mathcal{S}) = 0.99$. Note that, without using generalization and disjointness, we would have to settle for a lower bound of $\max_{p \in \mathcal{S}} cp(p) = 0.55$ and upper bound of $\min(1, \sum_{p \in \mathcal{S}} cp(p)) = 1$.*

8.2 Computing the Top-K Summary

We apply a best-first search approach to compute a top-k summary given a set of patterns $\textsc{Pat}_{lca}$. Our approach maintains a priority queue of candidate sets sorted on a lower bound $\underline{sc}$ for the score of candidate sets that we compute based on the completeness bound $\underline{cp}$ introduced above. We also maintain an upper bound $\overline{sc}$. For a set $\mathcal{S}$ of size k, we can compute $info(\mathcal{S})$ exactly. For incomplete candidates (size less than k), we bound the informativeness and completeness of any extension of the candidate into a set of size k using worst-case/best-case assumptions. For example, to bound completeness for an incomplete candidate $\mathcal{S}$ from above, we assume that the remaining patterns will not overlap with any pattern from $\mathcal{S}$ and have maximal completeness ($\max_{p \in \textsc{Pat}_{lca}} cp(p)$). We initialize the priority queue with all singleton subset of $\textsc{Pat}_{lca}$ and, then, repeatably take the incomplete candidate set with the highest $\underline{sc}$ and extend it by one pattern from $\textsc{Pat}_{lca}$ in all possible ways and insert these new candidates into the queue. The algorithm terminates when a complete candidate $\mathcal{S}_{best}$ is produced for which $\underline{sc}$ is higher than the highest $\overline{sc}$ value of all candidates we have produced so far (efficiently maintained using a max-heap sorted on $\overline{sc}$). In this case, we return $\mathcal{S}_{best}$ since it is guaranteed to have the highest score even though we do not know the exact value. The algorithm also terminates when all candidates have been produced, but no $\mathcal{S}_{best}$ has been found. In this case, we apply the following heuristic: we return the set with the highest average $((\underline{sc} + \overline{sc})/2)$.

9. EXPERIMENTS

We evaluate (i) the performance of computing summaries and (ii) the quality of summaries produced by our technique.

Experimental Setup. All experiments were executed on a machine with 2 x 3.3Ghz AMD Opteron CPUs (12 cores)

[4]Note that this is the intractable weighted maximal clique problem. For reasonably small k we can solve the problem exactly and otherwise apply a greedy heuristic.

$$r_1 : \texttt{InvalidD}(C) :- \texttt{LICENSE}(I, B, G, C, T, \mathsf{d}), \neg\texttt{VALID}(I)$$

$$r_2 : \texttt{Fsenior}(C) :- \texttt{LICENSE}(I, B, \mathsf{f}, C, T, L), \texttt{VALID}(I), B < 1953$$

$$r_3 : \texttt{CasualWatch}(T, E, N) :- \texttt{MOVIES}(I, T, Y, R, P, B, V), \texttt{GENRES}(I, E), \texttt{PRODCOMPANY}(I, C), \texttt{COMPANY}(C, N), \texttt{RATINGS}(U, I, G, S), \neg\texttt{GENRES}(I, \mathsf{thriller}), R < 100, G >= 4$$

$$r_4 : \texttt{Players}(A) :- \texttt{MOVIES}(I, T, Y, R, P, B, V), \texttt{CASTS}(I, C, H, A, G), \texttt{GENRES}(I, \mathsf{romance}), \texttt{RATINGS}(U, I, N, S), Y > 1999, N >= 4$$

$$r'_4 : \texttt{Players}(A) :- \texttt{MOVIES}(I, T, Y, R, P, B, V), \texttt{CASTS}(I, C, H, A, G), \texttt{GENRES}(I, \mathsf{comedy}), \texttt{KEYWORDS}(I, \mathsf{love}), \texttt{RATINGS}(U, I, N, S), Y > 1999, N >= 4$$

$$r''_4 : \texttt{Players}(A) :- \texttt{MOVIES}(I, T, Y, R, P, B, V), \texttt{CASTS}(I, C, H, A, G), \texttt{GENRES}(I, \mathsf{drama}), \texttt{KEYWORDS}(I, \mathsf{relationship}), \texttt{RATINGS}(U, I, N, S), Y > 1999, N >= 4$$

$$r_5 : \texttt{CommCrime}(T) :- \texttt{CRIMES}(I, Y, T, L, \mathsf{austin}), \neg\texttt{ARREST}(I)$$

$$r_6 : \texttt{CrimeSince}(T) :- \texttt{CRIMES}(I, Y, T, L, C), \neg\texttt{ARREST}(I), Y > 2012$$

$$r_7 : \texttt{FavCom}(T) :- \texttt{MOVIES}(I, T, Y), \texttt{GENRES}(I, \mathsf{comedy}), \texttt{RATES}(U, I, R, M, A), R \geq 4$$

$$r_8 : \texttt{ActMov}(T) :- \texttt{MOVIES}(I, T, Y), \texttt{GENRES}(I, \mathsf{action}), \texttt{RATES}(U, I, 5, M, A)$$

$$r_9 : \texttt{Hops}(L) :- \texttt{DBLP}(L, R), \texttt{DBLP}(R, R1), \texttt{DBLP}(R1, R2), \texttt{DBLP}(R2, R3), \texttt{DBLP}(R3, R4), \texttt{DBLP}(R4, R5)$$

$$r_{10} : \texttt{Custs}(CN, NK) :- \texttt{CUSTOMER}(CK, CN, C_1, NK, C_2, C_3, C_4, C_5), \texttt{ORDERS}(OK, CK, O_1, O_2, O_3, O_4, O_5, O_6, O_7), \texttt{LINEITEM}(OK, L_1, L_2, L_3, \cdots, L_{13}, L_{14}, L_{15})$$

$$r_{11} : \texttt{DirGen}(N) :- \texttt{MOVIES}(I, T, Y, R, P, B, V), \texttt{CREWS}(I, W, N, \mathsf{director}, M), \texttt{GENRES}(I, E), B > 20000000$$

$$r_{12} : \texttt{TomKey}(T, K, E) :- \texttt{MOVIES}(I, T, Y, R, P, B, V), \texttt{CASTS}(I, C, H, \mathsf{tom\ cruise}, G), \texttt{KEYWORDS}(I, K), \texttt{GENRES}(I, E), \texttt{RATINGS}(U, I, A, S), A \geq 4$$

Figure 4: Queries used in the experiments

and 128GB RAM running Oracle Linux 6.4. We use a commercial DBMS (name omitted due to licensing restrictions).

Datasets. We use TPC-H and several real-world datasets: (i) the New York State (NYS) license dataset[5] ($\sim$ 16M tuples), and (ii) a movie dataset[6] ($\sim$ 26M tuples), (iii) a Chicago crime dataset[7] ($\sim$ 6M tuples), and (iv) a co-author graph relation extracted from DBLP[8]. For each dataset, we created several subsets; $\mathtt{R_x}$ denotes a subset of $\mathtt{R}$ with x rows.

Queries. Fig. 4 shows the queries used in the experiments. For the license dataset, we use `InvalidD` (r_1) which returns cities with invalid driver's licenses and `Fsenior` (r_2) which returns cities with valid licenses held by female seniors. For the movie dataset, `CasualWatch` (r_3) returns movies with their genres and production companies if the runtime is less than 100 minutes and they have received high ratings ($G \geq 4$). `Players` (r_4) computes actresses/actors who have been successful in a romantic comedy after 1999 (defined as having received a rating higher than 4). In addition, `DirGen` (r_{11}) computes name of person who has directed a movie that has a budget over 2M dollars, and `TomKey` (r_{12}) returns movie title, keyword, and genre that Tom Cruise has played and has gotten. For the crime dataset, `CommCrime` (r_6) and `CrimeSince` (r_7) return types of unarrested crimes in the community Austin and anywhere since 2012, respectively. For the DBLP dataset, `Hops` (r_8) returns authors that are connected to each other by a path of length 6 in the co-author graph. For TPC-H, `Custs` (r_9) returns ids and the nations of customers who have at least one order.

[5]https://data.ny.gov/Transportation/Driver-License-Permit-and-Non-Driver-Identificatio/a4s2-d9tt

[6]https://www.kaggle.com/rounakbanik/the-movies-dataset

[7]https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-present/ijzp-q8t2

[8]http://www.dblp.org

Q	Why	Why-not	Q	Why	Why-not
r_1	new york	swanton	r_7	forrest gump	babysitting
r_2	brooklyn	delaware	r_8	fight club	avalanche
r_3	E = drama	E = family	r_9	-	xueni pan
r_4	jack black	tom ford	r_{10}	-	various
r_5	battery	domestic violence	r_{11}	steven spielberg	robert altman
r_6	theft	ritualism	r_{12}	K = mission	K = spying

Figure 5: Why and why-not provenance questions used in the experiments.

9.1 Performance

We consider samples of varying size n_S (`Sx` denotes a sample with x rows). Furthermore, `Full` denotes using the full provenance as input to the summarization process. Unless indicated otherwise, we use a 30 minute timeout for each experiment. Missing bars indicate timed-out experiments.

Dataset Size. We measure the runtime of our approach for computing top-3 summaries varying dataset and sample size over the queries in Fig. 4. On the x-axis of plots we show both the dataset size (#rows, lower part) and provenance size (#derivations, upper part). In Fig. 6a and 6b, we show the runtime for the individual steps of our algorithm (sampling, pattern generation, computation of quality metrics, and computing the top-3 summary) for query r_1 when binding C to new york and swanton, respectively (Fig. 5). Observe that, even for the largest dataset, we are able to generate summaries within reasonable time if using sampling. Overall, pattern generation dominates the runtime for why provenance. For queries r_3 (many joins with a negation) and r_4 (union of r'_4 and r''_4), the runtimes that follow similar trend are shown in Fig. 6c and 6e, respectively. For why-not provenance, sampling dominates the runtime for smaller sample sizes while pattern generation is dominant for `S10K`. `FULL` does not finish even for the 1K dataset. The runtimes for why-not provenance using r_3 and r_4 are shown in Fig. 6d and 6f, respectively. We observe the same trend as for r_1 even though why-not provenance is significantly larger (up to 10^{52} derivations). Fig. 7 shows more runtimes for both why and why-not provenance using queries r_2, r_{11}, and r_{12}. The bindings for provenance questions are shown in Fig. 5. We observe that these queries follow same trend as others on both why and why-not provenance.

Performance Comparison with Naive Approach. We also compare the performance of our sample-based summaries to the summary over a `FULL` set of derivations. The `FULL` is shown as '+' or red bars in Fig. 6 and 7. The results of `FULL` for why provenance are quadratic increase over the size of successful derivations while summaries result in almost linear increase. Computing summaries over `FULL` why-not provenance are not feasible within the allocated time slot for any size (bars are omitted in Fig. 6 and 7).

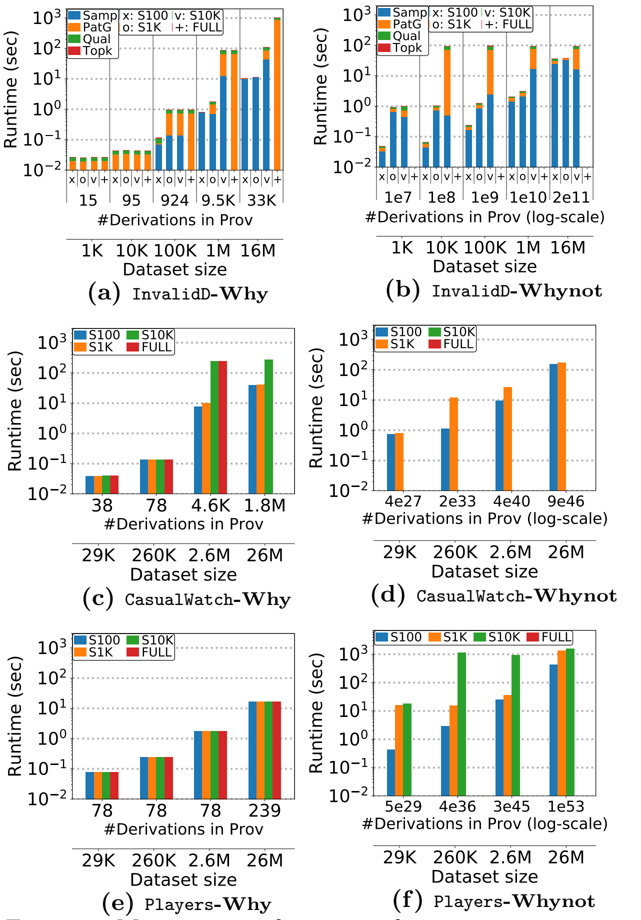


Figure 6: Measuring performance of generating summaries for why and why-not provenance for queries r_1, r_3, and r_4

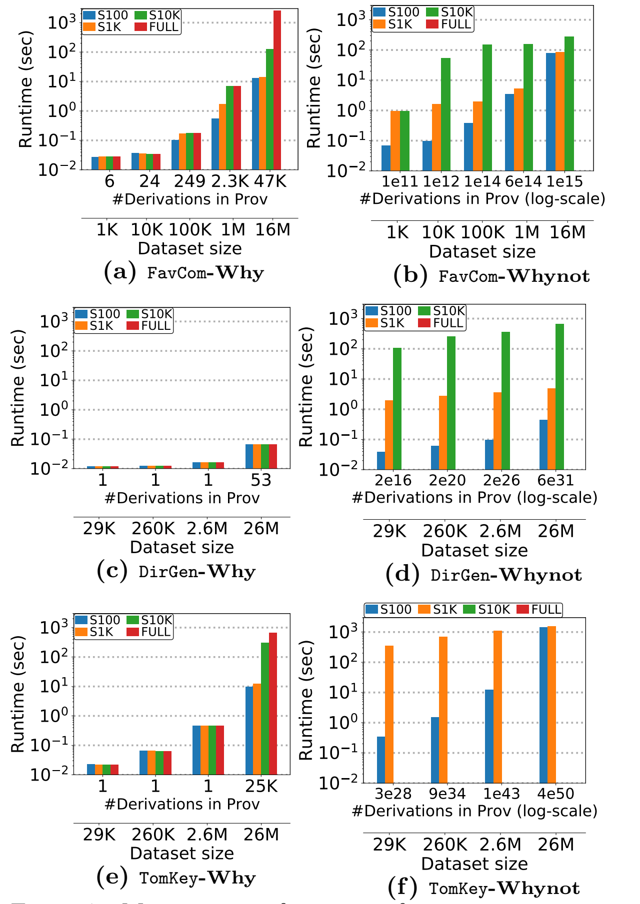


Figure 7: Measuring performance of generating summaries for why and why-not provenance for queries r_2, r_{11}, and r_{12}

Generating Top-k summaries. We now vary k from 1 to 10. Fig. 8 shows the runtime of computing the top-k summary over the patterns produced by the first three steps. The runtime of this step mainly depends on the number of patterns in its input. Overall, we observe that the choice of k does not add significant impact on performance. The computation takes up to 100sec for a complex query (e.g., r_4) and the largest sample (S10K) over $\texttt{MOVIE}_{2.6M}$.

Query Complexity and Structure. In this experiment, we vary the query complexity in terms of number of joins and number of variables. We compute the top-3 patterns for why-not questions. For this experiment, we randomly generated synthetic queries by varing those parameters. The join graph of these queries is either a star or a chain. The queries are evaluated over synthetic datasets with 100K tuples each. Fig. 9a and 9b show the runtime when varying the number of joins. The results confirm that our approach scales to very large provenance sizes (more than 10^{60} derivations) regardless of join types. We now evaluate the impact of the number of variables on performance. Here, we consider 8-way joins for chain queries and 5-way joins for star queries and vary the number of bindings from 1 to 16. Note that the head attribute and join attributes are not bound. The results shown in Fig. 9c and 9d confirm that our approach works well, even for queries with up to 24 variables (and provenance sizes of up to $\sim 10^{62}$ derivations). We now extend the evaluation with other queries and datasets. To extend the evaluation for variable impacts, we use query r_{10} (the join size is fixed to 3) over TPC-H_{150K} and compute summaries of why-not provenance. By binding an increasing number of variables from r_{10} to constants, we generate 6 rules that contain between 5 and 29 existential variables. The result shown in Fig. 9f confirms that our approach still scales to extreme provenance sizes (up to 10^{80} derivations). We also extend the experiment for join size with real-word dataset. Using the DBLP_{100K} dataset, we vary the number of joins (path length) of query r_9. For example, $\texttt{2Hop}(L) :- \texttt{DBLP}(L, R), \texttt{DBLP}(R, R1)$ is the query we use for a 2-way join. We use a p-tuple that binds $L = \texttt{xueni pan}$. Fig. 9e shows that even for real-world dataset with a 6-way joins where the provenance contains $3 \cdot 10^{26}$ derivations, we produce a result for sample sizes S100 and S1K.

9.2 Pattern Quality

We now measure the relative error introduced by sampling, i.e., the difference between the approximated quality metrics produced by sampling and the exact values when using the full provenance. For why-not provenance where it is not feasible to compute full provenance we compare against the largest sample size instead.

Quality Metric Error. Fig. 10c and 10d show relative quality metric error for query r_1 over $\texttt{InvalidD}_{100K}$ varying sample size and k. The error is at most $\sim 2\%$ and typi-

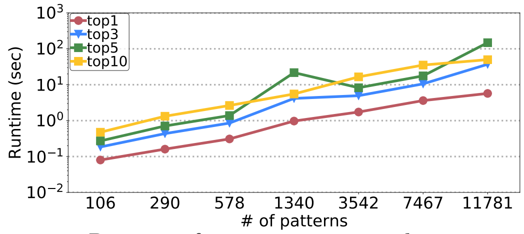
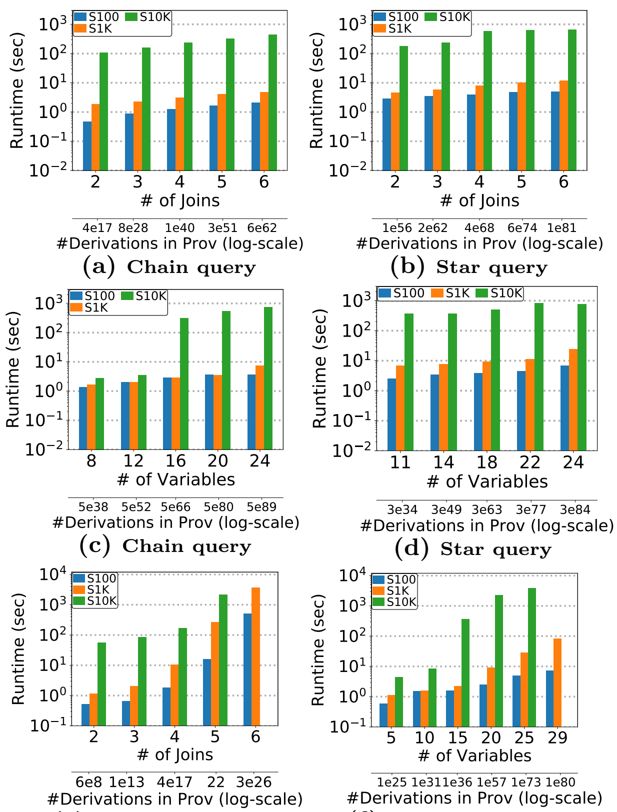
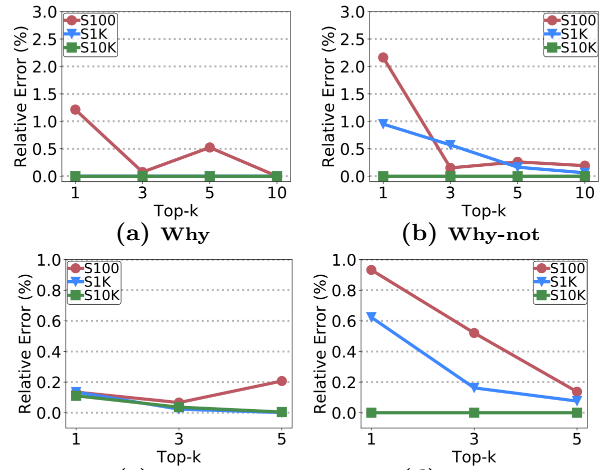


Figure 8: Runtime for computing top-k summaries when patterns are provided as input.



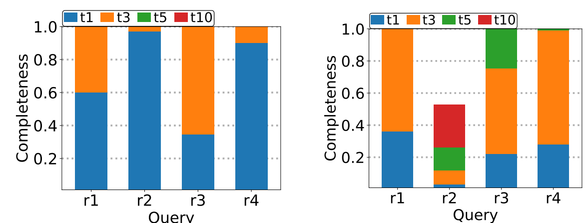
(a) **Chain query** (b) **Star query** (c) **Chain query** (d) **Star query** (e) r_7 **over DBLP$_{100K}$** (f) r_8 **over TPC-H$_{150K}$**

Figure 9: Varying #joins and #variables for why-not



(a) **Why** (b) **Why-not** (c) **Why** (d) **Why-not**

Figure 10: Quality metric error caused by sampling.



(a) **Why** (b) **Why-not**

Figure 11: Completeness - varying k.

cally decreases in k. We also measure for query r_6 over $\texttt{Crimes}_{1M}$ varying sample size and k. Fig. 10c Fig. 10d show results. Overall the relative error caused by sampling is still quite low (below 1%) and descreases in k and sample size.

Summary Completeness. Fig. 11a and 11b show the total completeness achieved by summaries returned by our approach for queries from Fig. 4. We calculate this by measuring the fraction of provenance covered by at least one pattern from a summary. Observe that completeness increases for larger values of k and that for both why and why-not we achieve $\sim$ 100% completeness with only $k = 5$ except why-not questions over r_2 (Fig. 11b). Although the unbound attributes for r_2 have a large number of distinct values, the top-10 summary covers the majority of provenance.

9.3 Comparisons with other systems

We now compare computing summaries with our system (*PUG-Summ*) against *Artemis* [13] (all-derivations) and a single-derivation approach implemented in our system.

Artemis. The authors of [13] made their system available as a virtual machine. We ran both systems in this virtual machine (4GB memory). For this particular evaluation, we use Postgres as a backend since it is supported by both PUG and Artemis. We choose one of the queries that was available in the VM installation, shown in Datalog below.

$$\texttt{CrimeDesc}(T,N,C,H) :- \texttt{CRIME}(T,S), \texttt{WITNESS}(N,S), \texttt{SAWPERSON}(N,H,C), \texttt{PERSON}(M,H,C), S > 97$$

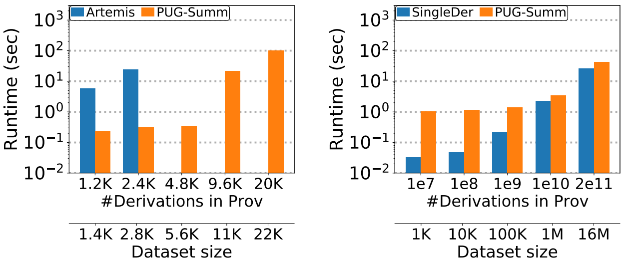
This query computes names of witnesses (N) that saw a person with particular cloth and hair color perpetrating a crime of a particular type T. We use the provenance question provided by Artemis: T = 'trespassing', N = 'Aarongolden', C = 'MidnightBlue', and H = 'lavender'. The original dataset is $\texttt{CRIME}_{1.4K}$ which we scaled up to $\texttt{CRIME}_{22K}$. We use $\sim$ 10% as the sample size for each version of the dataset (e.g., $\texttt{S2K}$ for $\texttt{CRIME}_{22K}$) and compute top-5 summaries. The result of this comparison is shown in Fig. 12a. Our system outperforms Artemis on all datasets. Artemis was only able to return explanations for very small datasets while our system computes summaries for all datasets within seconds. Artemis returned the most general pattern (all placeholders except the provided constants) as the top-1 explanation:

$$p = (\mathsf{trespassing}, \mathsf{Aarongolden}, \mathsf{MidnightBlue}, \mathsf{lavender}, S, M), \quad S > 97$$

Unlike Artemis, PUG returned a summary that contains a pattern which covers $\sim$ 50% of the provenance:

$$p' = (\mathsf{trespassing}, \mathsf{Aarongolden}, \mathsf{MidnightBlue}, \mathsf{lavender}, 98, M)$$

Single Derivation Approach. For this comparison, we implemented a simple single-derivation approach. We use



(a) Artemis **(b) Single-derivation**

Figure 12: Performance comparisons for why-not

query r_1 from Fig. 4, a sample size of S1K, and compute a top-3 summaries. Fig. 12b shows the results of this comparison. For small datasets, the single derivation approach outperforms PUG's summarization about an order of magnitude. The gap between the two approaches is less significant for larger datasets.

10. RELATED WORK

Compact Representation of Provenance. Provenance models for database queries have been studied extensively [8, 16]. The need for compressing provenance to reduce its size has been also recognized early-on, e.g., [3, 7, 24]. However, the compressed representations produced by these approaches are often not semantically meaningful to users. More closely related to our work are techniques for generating higher-level explanations for binary outcomes [9, 31], missing answers [28], or query results [26, 32, 2] as well as methods for summarizing data or general annotations which may or may not encode provenance information [35]. Specifically, like [9, 31, 26, 32] we use patterns with placeholders. Some approaches use ontologies [28, 31] or logical constraints [26, 9, 32] to derive semantically meaningful and compact representations of a set of tuples. The use of constraints to compactly represent large or even infinite database instances has a long tradition [15, 18] and these techniques have been adopted to compactly explain missing answers [13, 25]. However, the compactness of these representations comes at the cost of computational intractability.

Missing Answers. The missing answer problem was first stated for query-based explanations (which parts of the query are responsible for the failure to derive the missing answer) in the seminal paper by Chapman et al. [6]. Most follow-up work [4, 5, 6, 29] is based on this notion. Huang et al. [14] first introduced an instance-based approach, i.e., which existing and missing input tuples caused the missing answer [13, 14, 20, 22]). Since then, several techniques have been developed to exclude spurious explanations and to support larger classes of queries [13]. As mentioned before, approaches for instance-based explanations use either the all-derivations (giving up performance) or the single-derivation approach (giving up completeness). In contrast, using summarizes we guarantee performance by compactly representing large amounts of provenance without forsaking completeness. Similarly, some missing answer approaches [13] use c-tables to compactly represent sets of missing answers. However, this comes at the cost of additional computational complexity and necessitates the use of constraint solvers.

11. CONCLUSIONS

We have presented an approach for efficiently computing summaries of why and why-not provenance that are compact and descriptive. Our approach uses sampling to generate summaries that are guaranteed to be concise while balancing completeness (the fraction of provenance covered) and informativeness (new information provided by the summary). Thus, we overcome a severe limitation of prior work which sacrifices either completeness or performance. We demonstrate experimentally that our approach can efficiently produce meaningful summaries of very large provenance graphs (up to 10^{80} derivations). In future work, we plan to investigate how additional information, e.g., integrity constraints, can be utilized in the summarization process.

12. REFERENCES

[1] M. Abramowitz and I. A. Stegun. *Handbook of mathematical functions: with formulas, graphs, and mathematical tables*, volume 55. Courier Corporation, 1965.

[2] E. Ainy, P. Bourhis, S. Davidson, D. Deutch, and T. Milo. Approximated summarization of data provenance. In *CIKM*, pages 483–492, 2015.

[3] M. K. Anand, S. Bowers, T. McPhillips, and B. Ludäscher. Efficient Provenance Storage over Nested Data Collections. In *EDBT*, pages 958–969, 2009.

[4] N. Bidoit, M. Herschel, and K. Tzompanaki. Immutably answering why-not questions for equivalent conjunctive queries. In *TaPP*, 2014.

[5] N. Bidoit, M. Herschel, K. Tzompanaki, et al. Query-Based Why-Not Provenance with NedExplain. In *EDBT*, pages 145–156, 2014.

[6] A. Chapman and H. V. Jagadish. Why Not? In *SIGMOD*, pages 523–534, 2009.

[7] A. Chapman, H. V. Jagadish, and P. Ramanan. Efficient Provenance Storage. In *SIGMOD*, pages 993–1006, 2008.

[8] J. Cheney, L. Chiticariu, and W. Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1(4):379–474, 2009.

[9] K. El Gebaly, P. Agrawal, L. Golab, F. Korn, and D. Srivastava. Interpretable and informative explanations of outcomes. *PVLDB*, 8(1):61–72, 2014.

[10] K. E. Gebaly, G. Feng, L. Golab, F. Korn, and D. Srivastava. Explanation tables. *IEEE Data Eng. Bull.*, 41(3):43–51, 2018.

[11] T. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, pages 31–40, 2007.

[12] M. Herschel, R. Diestelkämper, and H. Ben Lahmar. A survey on provenance: What for? what form? what from? *VLDB J.*, 26(6):881–906, 2017.

[13] M. Herschel and M. Hernandez. Explaining Missing Answers to SPJUA Queries. *PVLDB*, 3(1):185–196, 2010.

[14] J. Huang, T. Chen, A. Doan, and J. Naughton. On the provenance of non-answers to queries over extracted data. In *VLDB*, pages 736–747, 2008.

[15] T. Imieliński and W. Lipski Jr. Incomplete Information in Relational Databases. *JACM*, 31(4):761–791, 1984.

[16] G. Karvounarakis and T. J. Green. Semiring-annotated data: queries and provenance. *SIGMOD Record*, 41(3):5–14, 2012.

[17] S. Köhler, B. Ludäscher, and D. Zinn. First-order provenance games. In *In Search of Elegance in the Theory and Practice of Computation*, pages 382–399. Springer, 2013.

[18] G. M. Kuper, L. Libkin, and J. Paredaens, editors. *Constraint Databases*. Springer, 2000.

[19] S. Lee, , B. Ludäscher, and B. Glavic. Computing why and why-not provenance summaries with sampling. Technical Report IIT/CS-DB-2019-01, http://cs.iit.edu/%7edbgroup/assets/pdfpubls/LLG19.pdf, Illinois Institute of Technology, 2019.

[20] S. Lee, S. Köhler, B. Ludäscher, and B. Glavic. A SQL-middleware unifying why and why-not provenance for first-order queries. In *ICDE*, pages 485–496, 2017.

[21] S. Lee, B. Ludäscher, and B. Glavic. Provenance summaries for answers and non-answers. In *PVLDB (Demo)*, pages 1954–1957, 2018.

[22] S. Lee, B. Ludäscher, and B. Glavic. Pug: a framework and practical implementation for why and why-not provenance. *VLDB J.*, pages 1–25, 2018.

[23] S. Lee, X. Niu, B. Ludäscher, and B. Glavic. Integrating approximate summarization with provenance capture. In *TaPP*, pages 2–2, 2017.

[24] D. Olteanu and J. Závodný. On factorisation of provenance polynomials. In *TaPP*, 2011.

[25] S. Riddle, S. Köhler, and B. Ludäscher. Towards constraint provenance games. In *TaPP*, 2014.

[26] S. Roy and D. Suciu. A formal approach to finding explanations for database queries. In *SIGMOD*, pages 1579–1590, 2014.

[27] V. Tannen. Provenance analysis for FOL model checking. *ACM SIGLOG News*, 4(1):24–36, 2017.

[28] B. ten Cate, C. Civili, E. Sherkhonov, and W.-C. Tan. High-level why-not explanations using ontologies. In *PODS*, pages 31–43, 2015.

[29] Q. T. Tran and C.-Y. Chan. How to conquer why-not questions. In *SIGMOD*, pages 15–26, 2010.

[30] E. Von Collani and K. Dräger. *Binomial distribution handbook for scientists and engineers*. Springer Science & Business Media, 2001.

[31] X. Wang, X. L. Dong, and A. Meliou. Data x-ray: A diagnostic tool for data errors. In *SIGMOD*, pages 1231–1245, 2015.

[32] E. Wu and S. Madden. Scorpion: Explaining away outliers in aggregate queries. *PVLDB*, 6(8):553–564, 2013.

[33] Y. Wu, A. Haeberlen, W. Zhou, and B. T. Loo. Answering why-not queries in software-defined networks with negative provenance. In *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, page 3. ACM, 2013.

[34] Y. Wu, M. Zhao, A. Haeberlen, W. Zhou, and B. T. Loo. Diagnosing missing events in distributed systems with negative provenance. In *SIGCOMM*, pages 383–394, 2014.

[35] D. Xiao and M. Y. Eltabakh. Insightnotes: Summary-based annotation management in relational databases. In *SIGMOD*, pages 661–672, 2014.