Integrating Approximate Summarization with Provenance Capture

Seokki Lee[1] Xing Niu[1] Bertram Ludäscher[2] Boris Glavic[1]

[1]Illinois Institute of Technology
{slee195, xniu7}@hawk.iit.edu, bglavic@iit.edu

[2]University of Illinois at Urbana-Champaign
ludaesch@illinois.edu

Abstract

How to use provenance to explain why a query returns a result or why a result is missing has been studied extensively. Recently, we have demonstrated how to uniformly answer these types of provenance questions for first-order queries with negation and have presented an implementation of this approach in our *PUG* (*P*rovenance *U*nification through *G*raphs) system. However, for realistically-sized databases, the provenance of answers and missing answers can be very large, overwhelming the user with too much information and wasting computational resources. In this paper, we introduce an (approximate) *summarization* technique that generates compact representations of why and why-not provenance. Our technique uses patterns as a summarized representation of sets of elements from the provenance, i.e., successful or failed derivations. We rank these patterns based on their descriptiveness (we use precision and recall as quality measures for patterns) and return only the top-k summaries. We demonstrate how this summarization technique can be integrated with provenance capture to compute summaries on demand and how sampling techniques can be employed to speed up both the summarization and capture steps. Our preliminary experiments demonstrate that this summarization technique scales to large instances of a real-world dataset.

1. Introduction

Provenance for relational queries [GKT07] explains how results of a query depend on the query's inputs. Recently, provenance-like techniques have been applied to explain how missing inputs cause a tuple to be missing from the query result (e.g., [HH10, MGMS10]). In prior work, we have shown that why and why-not questions can be treated uniformly as provenance for first-order (FO) queries, encoded as non-recursive Datalog with negation [KLZ13], and have presented an implementation called *PUG (Provenance Unification through Graphs)* [LKLG17] which runs on top of a relational database backend. Typically, only a part of the provenance (which we call an *explanation*) is relevant to answer a provenance question about the existence or absence of a query result. An explanation for either a why or why-not question should justify the existence or absence of a result based on success or failure to derive the result using the rules of the input query. Moreover, it should also explain how the existence or absence of tuples in the database caused the derivation to succeed or fail, respectively. Provenance graphs generated by PUG provide this type of justification for a (missing) query answer [LKLG17]. Like *SelP* [DGM15], PUG generates explanations by instrumenting input queries to capture relevant provenance. The main difference between this approach and PUG is that we support negation and why-not (which SelP does not support), but do not support recursion yet.

r_0 : $\texttt{LTCRIME}(LT, T) :- \texttt{CRIMES}(I, T, L, LT, C), \neg\texttt{ARREST}(I).$

CRIMES (input)

Id	Type	Location	LocType	Community
2446	assault	parking lot	public	new city
3465	burglary	apartment	private	south shore
1066	assault	street	public	new city
2415	burglary	residence	private	south shore
3645	theft	appliance store	public	lincoln park

ARREST (input)

Id
3465
2415

LTCRIME (output)

LocType	Type
public	assault
public	theft

Figure 1: Crime database, query r_0, and query result (LTCRIME)

EXAMPLE 1. *Consider the sample of a real Chicago crime dataset*[1] *shown in Fig. 1. The* CRIMES *table records for each crime an* id*, the crime* type *and* location *(e.g., assault at a street), the type of* location *(private or public), and the* community *in which the crime took place. Relation* ARREST *stores the* ids *of crimes that led to an arrest. Datalog query* r_0 *returns types of unsolved crimes for each location type, i.e., at least one crime of this type at this particular location type (private or public place) has not led to an arrest. The result over the sample instance is shown as* LTCRIME*. Given such a query, a user may be interested in understanding why a particular result is returned, e.g.,* why(public,assault)*, or why a tuple is missing from the result, e.g.,* whynot(private,burglary)*. The explanation generated by PUG for* whynot(private,burglary) *enumerates all potential derivations of this fact using the rule* r_0 *and for each such derivation provides the justification for why it failed. Fig. 2 shows part of the provenance graph (explanation) for this question. We will discuss these types of graphs in more depth in Sec. 2. The fragment shown in Fig. 2 represents one failed binding of rule* r_0 *with id* 3465*, location* apartment*, and community* south shore*. This rule derivation failed because, while there was a burglary with id* 3465 *in an apartment in community* south shore *(first*

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

TaPP 2017, June 22-23, 2017, Seattle, Washington.

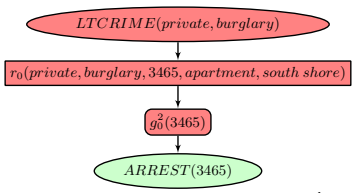

Figure 2: Partial provenance graph explaining (private,burglary)

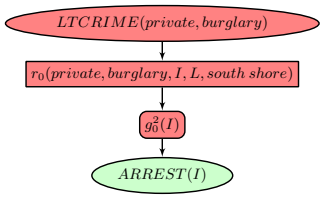

Figure 3: Summarized explanation for why-not (private,burglary)

goal succeeds), the offender was arrested (goal $\neg$ ARREST(*3465*) *failed).*

Summarizing Provenance. While explanations are useful for limiting the scope of provenance to a (missing) result of interest, the resulting provenance graphs may still be very large, in particular for missing answers where we have to enumerate all potential ways of how a result could have been derived and for each such alternative have to explain why it failed. For instance, even for the toy example from Fig. 1, the full provenance graph for whynot(private,burglary) already contains 75 failed rule derivations corresponding to all combinations of ids, locations, and communities ($5 \cdot 5 \cdot 3$). For the full dataset of $\sim$ 8 million records, there are $\sim$ 71 million failed derivations. However, if we also take into account that the full dataset contains additional attributes that we have not shown here, then there are $\sim 3.3 \cdot 10^{42}$ such failed derivations. Motivated by this problem, we present an extension of PUG which summarizes provenance graphs to create compact, human-readable explanations. The need for compressing provenance to reduce its size has been recognized early-on, e.g., [ABML09, CJR08, OZ11]. However, the compressed representations produced by these approaches are often not semantically meaningful to a human. More closely related to our work are techniques for generating higher-level explanations for binary outcomes [EGAG+14, WDM15], missing answers [tCCST15], or query results [RS14, WM13, ADDM14] as well as methods for summarizing data or general annotations which may or may not encode provenance information [XE14]. Some approaches use ontologies [tCCST15, WDM15] or logical constraints [RS14, EGAG+14, WM13] to derive semantically meaningful and compact representations of a set of tuples to be described. Sometimes such constraints are represented as queries or tuple patterns with placeholders. For instance, if a geographical taxonomy is available then the set of all cities in Illinois (probably 100s or 1000s) can be compactly represented using a single concept `IllinoisCity` from the taxonomy. The use of constraints to compactly represent large or even infinite database instances has a long tradition [ILJ84, KLP00] and these techniques have been adopted to compactly explain missing answers [HH10, RKL14]. However, the compactness of these representations comes at the cost of computational intractability. Similar to some of these approaches, we also use patterns that encode constraints to summarize provenance.

EXAMPLE 2. *We produce* summarized explanations *by concisely encoding sets of nodes in provenance graphs as* pattern nodes*. The label of a pattern node contains a mix of constants and variables (placeholders). A pattern node* p *represents all nodes that can be derived from* p *by substituting suitable constants for* p*'s variables. A summary for* whynot*(private,burglary) is shown in Fig. 3. The graph contains a pattern node* $p_1 = r_0$*(private,burglary,I,L,south shore) where* I *and* L *are placeholders. Thus,* p_1 *represents all nodes* $r_0(\mathsf{private}, \mathsf{burglary}, \mathsf{c}_1, \mathsf{c}_2, \mathsf{southshore})$ *for any crime id* c_1 *and location* c_2*. Using provenance graphs with pattern nodes, we can compactly represent large explanations. Note that some of the nodes represented by a pattern node may not belong to an explanation. We will address this potential issue in Sec. 3.*

The summarization technique described in the example above is similar to the pattern-based techniques from related work that we have mentioned above. It produces semantically meaningful summaries and can reduce the size of provenance significantly as long as the data has inherent hierarchical structure that can be exploited. Importantly, many datasets exhibit this type of hierarchical structure. Typical examples are geographical data (countries consist of states which contain cities and cities contain communities), temporal data (e.g., a year consists of 12 months), product data (products belong to categories), and biological datasets (e.g., species belong to families). In this work, we assume that the structure is explicitly represented in the data, e.g., temporal data should be represented using separate fields for separate levels of the time taxonomy. An interesting avenue for future work is how to automatically extract hierarchy information from data when it is not explicitly represented. Even if the data exhibits structure that we can exploit, summarization has to balance conflicting goals. Ideally, we want *summarized explanations* to be *concise* (small provenance graphs), *correct* (the patterns only cover provenance), and *complete* (the summary covers the full provenance). Not surprisingly, this is often not possible. For instance, we can trivially achieve completeness and correctness by returning the full explanation for a user question, but this completely abandons conciseness. We decided to implement a solution which ranks of candidate pattern rule nodes based on a combined score derived from their relative completeness and correctness. Given a provenance question, we return a single summarized explanation that consists of the top-k rule pattern nodes according to this ranking and directly connected goal and tuple nodes. Thus, we guarantee conciseness by bounding the size of summarized explanations. In Sec. 3, we discuss pattern-based summary model, pattern candidate generation, and ranking.

Approximate Summarization with Sampling. Using summarization, we can address the usability issues of large provenance graphs. However, producing a full provenance graph for a user question as input for summarization is computationally infeasible. No matter how efficient provenance capture is, generating graphs with 10^{42} nodes is not practical. To be able to generate provenance-based explanations over realistically-sized databases, we introduce sampling techniques that are applied during provenance capture. Our preliminary experimental evaluation shows that our sampling method only introduces a small amount of error into the measures that we use for ranking. We leave a rigorous probabilistic analysis of error bounds for future work.

Contributions. To the best of our knowledge, we are the first to address both the usability and computational challenges of why and why-not provenance by producing semantically meaningful summaries based on approximate provenance generated by integrating sampling into provenance capture. Specifically, we make the following contributions:

- We use patterns to generate semantically meaningful summarized explanations for both why and why-not questions
- We integrate provenance capture with summarization
- We employ sampling during provenance capture to avoid having to pay the price of materializing full provenance as the input to summarization

2. PUG (Provenance Unification through Graphs)

Our PUG system [LKLG17] computes the explanation for user's provenance question about the existence or absence of a query result as a provenance graph[2] that encodes how successful and failed derivations of a fact using an input query's rules have lead to the existence or absence of the result. Furthermore, for each rule derivation included in an explanation, these graphs provide the justification for why it succeeded (all its goals evaluated to true) or failed (some of its goals evaluated to false).

Rule Derivations. PUG generates provenance graphs based on derivations of the rules of a query Q. A rule derivation is the result of applying a valuation, a mapping from the variables of a rule r to constants. Let $\bar{c}$ be a list of constants, one for each variable of a rule r. We use $r(\bar{c})$ to denote the derivation that assigns c_i to variable X_i in rule r. Note that we order variables by the position of their first occurrence in the rule, e.g., the variable order for r_0 (Fig. 1) is (LT, T, I, L, C). For a given database instance D, a rule derivation is successful if $D \models r(\bar{c})$, i.e., the goals of the rule evaluate to true given the database D. For failed rule derivations, we are interested in which goals are failed and, thus, justify the failure. Therefore, we associate a rule derivation with a list $\bar{g}$ of boolean values indicating for each goal whether it succeeds or not.

DEFINITION 1 (Annotated Rule Derivation). *Let D be a database,*

$$Q(\overline{x_h}) :- R_1(\overline{x_1}), \ldots, R_m(\overline{x_m}), \neg R_{m+1}(\overline{x_{m+1}}), \ldots, \neg R_n(\overline{x_n})$$

be a Datalog rule and $\overline{x} = \bigcup_{i=1}^{n} \overline{x_i}$. An annotated rule derivation is a pair of a list $\bar{c}$ of constants with the same arity as $\bar{x}$ and a list $\bar{g} = (g_1, \ldots, g_n)$ with $g_i \in \{T, F\}$ such that

$$g_i = \begin{cases} T & \text{if } i \leq m \wedge D \models R_i(\overline{c_i}) \text{ or } i > m \wedge D \not\models R_i(\overline{c_i}) \\ F & \text{otherwise} \end{cases}$$

Here $\overline{c_i}$ denotes the constants from $\bar{c}$ that correspond to $\overline{x_i}$.

In the following, we will use $r(\bar{c}) - (\bar{g})$ to denote an annotated derivation of rule r with constants $\bar{c}$ and boolean goal indicators $\bar{g}$. Furthermore, we use $\mathcal{A}(Q, D, r)$ to denote all annotated derivations of rule r from Q according to D.

EXAMPLE 3. *For example, one failed annotated derivation of missing answer* (private,burglary) *is*

$$r_0(\textit{private}, \textit{burglary}, \textit{3465}, \textit{apartment}, \textit{south shore}) - (T, F)$$

Provenance Graphs for FO-queries. As mentioned above, we employ a graph-based provenance model for first-order (FO) queries introduced in prior work [LKLG17]. Given a question about the existence (absence) of a result, our system generates an *explanation*, i.e., a subgraph of the full provenance graph for a query that contains only facts and rule derivations relevant for deriving (or failing to derive) the result. Our graphs consist of *rule nodes* (boxes with a rule-id and the constant arguments of a rule derivation), *goal nodes* (rounded boxes with a rule-id and the goal's position in the rule body), and *tuple nodes* (ovals). Nodes are either *green* (successful/existing) or *red* (failed/missing).

EXAMPLE 4. *Fig. 2 shows a part of the provenance graph that PUG generates to explain the missing answer* (private,burglary). *This part corresponds to our exemplary annotated derivation. It contains the missing answer and a node representing the failed rule derivation. The failed derivation is connected to all failed goals (recall that in this example only the second goal failed). The node representing the failed goal* $\neg$ ARREST(*3465*) *is connected to the existing tuple* ARREST(3465) *justifying the goal's failure.*

Capturing Provenance. PUG computes the explanation (provenance graph) for a (missing) answer using a rewriting of the input Datalog query. The instrumented query that generates the explanation is translated into SQL code by the system which can then be run on a standard DBMS backend. The program constructed by this rewriting consists of two parts. First, the rules of the input query are transformed to capture annotated rule derivations relevant for explaining the (missing) answer. We refer to rules instrumented in this fashion as *firing rules*. The second part of the rewritten program generates the edge relation of the provenance graph by generating a set of edges for each annotated rule derivation. For instance, for the failed derivation used in the Example 4, we would generate the three edges shown in Fig. 2. We refer the interested reader to [LKLG17] for details of this rewriting. For the purpose of this paper, the important take-away message is that the annotated rule derivations fully determine the graph to be generated. We will exploit this fact by applying summarization to sets of annotated rule derivations and, then, generate the edges of a provenance graph from these summarized rule derivations.

Size of explanations. As mentioned in the introduction, the graphs generated by PUG can be quite large. In particular, for missing answers, we have to enumerate all potential ways (rule derivations) of how the answer could have been derived and, for each such derivation, explain why it failed (enumerating the failed goals in the rule's body). Under the open world assumption, there would be infinitely many failed rule derivations. While under the closed world assumption there only exists a finite number of derivations, this number would still be very large. A typical assumption is to bound variables by the active domain $adom(D)$, i.e., all values that occur in some attribute(s) in the database D or occur as constants in the query. Since we can bound any of the existential variables of a Datalog rule to any value of the active domain, the number of derivations for a domain of size $|D|$ and rule with m existential variables would be $|D|^m$. Note that using all values of $adom(D)$ indistinguishably for all variables of a rule may lead to semantically meaningless derivations (e.g., using community south shore as a value for attribute Type). In PUG, we allow the user to associate a domain of values with each attribute and exclusively use values from these domains to form rule derivations [LKLG17]. This avoids the construction of meaningless derivations, but does not solve the size issue. For instance, the numbers given in the introduction have been calculated based on the assumption that for each attribute A the user has assigned the set of values of A as the domain of A.

3. Provenance Summaries

The large size of provenance-based explanations may be overwhelming for a user even if provenance is limited to what is relevant for a particular (missing) result. For example, explanation for the question whynot(private,bulglary) has more than 370 nodes in total (75 failed rule derivations). We address this problem by creating summaries of provenance graphs based on the structure that is present in the provenance. In particular, we compactly represent sets of subgraphs through graph patterns, i.e., graphs where the arguments used in a node label can be placeholders (variables). As we have already outlined in Sec. 2, our provenance graphs are generated from annotated rule derivations. Thus, developing pattern-based summarization techniques for these rule derivations automatically yields a method for summarizing provenance graphs. We can use the part of the instrumented program generated by PUG which computes the edge relation of the provenance graph and apply it to

[2] We support different types of provenance graphs that differ in informativeness. For instance, one type encodes provenance polynomials [GKT07], the most general form of provenance in the semiring framework, if restricted to positive queries.

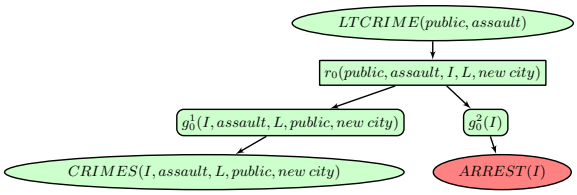

Figure 4: Summarized explanation for why (public,assault)

the rule derivation patterns generated by summarization to generate summarized provenance graphs.

Derivation Patterns. Rule derivation patterns are rule derivations which in addition to constants may contain placeholders.

DEFINITION 2 (Derivation Pattern). *Let D be a database and r a rule with n variables. A* derivation pattern *is a list $\bar{e}$ of length n where each $e_i \in adom(D) \cup \{*\}$. We use $r(\bar{e})$ to denote derivation pattern $\bar{e}$ for rule r.*

A derivation pattern represents the set of annotated rule derivations that *match* the pattern, i.e., all rule derivations that agree with the pattern's constants.

DEFINITION 3 (Pattern Matches). *Let D be a database, Q a query, and r a rule of Q. A derivation pattern $p = r(\bar{e})$ for a rule r with n variables matches an annotated rule derivation $d = r(\bar{c}) - (\bar{g}) \in \mathcal{A}(Q, D, r)$ written as $p \models d$ iff*

$$\forall i \in \{1, \ldots, n\} : e_i = * \vee e_i = c_i$$

We define $\mathcal{M}(p) = \{d \mid p \models d\}$ where Q and D are assumed to be clear from the context.

For readability, we will often use variables instead of $*$ in a derivation pattern to denote which variable of a rule a placeholders corresponds to.

EXAMPLE 5. *Continuing with Example 4, recall that*

$$r_0(\textit{private}, \textit{burglary}, \textit{3465}, \textit{apartment}, \textit{south shore}) - (T, F)$$

is one failed derivation of missing answer (private,burglary). *There may exist many other similar derivations which can be compactly encoded using derivation patterns. For instance, the pattern*

$$r_0(\textit{private}, \textit{burglary}, I, L, \textit{south shore})$$

represents all burglaries in south shore for any id and location.

Given derivation patterns, we generate provenance graphs corresponding to these patterns.

EXAMPLE 6. *A summarized explanation for* why(public,assault) *using patterns is shown in Fig. 4. The summary explains query result* (public,assault) *- there are assaults in public places (some public location L) in community new city which have not led to an arrest. That is, the pattern represents the set of all derivations using rule r_0 where community is new city, location is some location L, and the crime id is some I. Each derivation for a given $I = c_1$ and $L = c_2$ represented by the pattern corresponds to the subgraph from the provenance with the structure shown in Fig. 4.*

Generating Candidate Patterns. Given the rule derivations in the provenance of a query, we can generate candidate patterns by replacing any subset of the constants in a rule derivation with placeholders (variables). For instance, some of the patterns we can create from r_0(private,burglary,3465,apartment,south shore) are r_0(private,burglary,3465,L,south shore), r_0(private,burglary,I, apartment,C), and so on. Note that we do not allow head variables (LT and T in our example) to be replaced with placeholders since the binding for these variables are provided by the user question. If we apply this process exhaustively, then this leads to an exponential number of patterns (in the number of variables of rules). As observed in [EGAG+14], there are several ways of how to limit the number of generated patterns, e.g., by only creating patterns from a sample of the rule derivations. We will explain how we incorporate sampling into the summarization process in Sec. 4.

Selecting Top-k Patterns. The set of rule derivations $\mathcal{M}(p)$ represented by a pattern p for rule r may contain both derivations that belong to the provenance as well as derivations that are not in the provenance. To make this more concrete, we need to specify what is the set of rule derivations we consider for matching against a pattern and under which conditions is a rule derivation considered from this set to belong to the provenance. Given a why or why-not question $Q(\bar{c})$, we are only interested in derivations that agree with the constants of the user question. For example, for question why(public,assault), we are not interested in rule derivations that derivate another type of crime (e.g., a murder). That is, for evaluating how well a pattern summarizes provenance, we use a subset of $\mathcal{A}(Q, D, r)$ containing all derivations that agree with the constants of the user questions (bind head variables to these contants). Let $\mathcal{A}(Q(\bar{c}), r)$ denote this set for a question $Q(\bar{c})$. We partition this set into two subsets (provenance and non-provenance). For a why question $Q(\bar{c})$ and a rule r, all successful derivations of $Q(\bar{c})$ using rule r are considered as provenance and all failed derivations of r deriving $Q(\bar{c})$ as non-provenance. For a why-not question $Q(\bar{c})$, all derivations of the result using a rule r are failed. We summarize individual failure patterns independently since each failure pattern corresponds to a different subgraph structure in the provenance. For a failure pattern $\bar{g}$, we consider all rule derivations of $Q(\bar{c})$ as provenance if they have the same failure pattern and as non-provenance otherwise. Note that if a Datalog query consists of multiple rules, then not all rules in the provenance of a user question may directly derive the user question's tuple. This is dealt with by propagating constants from the user question to related rules (for a description of how we propagate constants, see [LKLG17]). The set of rule derivations is then determined based on the propagated constants. We use $Prov(p)$ to denote the number of rule derivations in the provenance matched by a pattern p, $NProv(p)$ to denote the number of derivations matched by pattern p that are not in the provenance, and $TProv$ to denote the total number of derivations in the provenance. We use this notation to define the precision $prec(p)$ and recall $rec(p)$ of a pattern (with respect to a provenance question) in the usual way:

$$prec(p) = \frac{Prov(p)}{Prov(p) + NProv(p)} \qquad rec(p) = \frac{Prov(p)}{TProv}$$

Note that precision effectively measures what we have called correctness in the introduction (the fraction of matched derivations that belong to the provenance) and recall measure completeness (the fraction of derivations in the provenance matched by the pattern). We use the F-measure (geometric mean of the precision and recall) to select the top-k pattern(s) for each type of rule derivation which provides concise summaries. In our experience, it is essential to consider both precision and recall in the ranking since ranking by precision alone often leads to overly specific patterns (e.g., a pattern without placeholders) and ranking on recall alone leads to very generic patterns (e.g., a pattern with only variables plus constants from the user question). Investigating other quality measures for patterns is an interesting avenue for future work.

EXAMPLE 7. *Recall that a summary for* whynot*(private,burglary) is shown in Fig. 3 and we summarize failed derivations independently for each failure pattern. Assume that the user requests the*

failure pattern (T, F)*, i.e., where only the second goal is failed. That is, there exists a burglary in a private place that has led to an arrest. The intuitive meaning of this summary is that all the burglaries in private locations led to arrests (the graph contains a pattern* $p_1 = r_0(\mathsf{private}, \mathsf{burglary}, I, L, \mathsf{south\ shore})$*). Pattern* p_1 *subsumes all derivations where the first goal succeeded (an existing burglaries), but the second goal failed (offender was arrested). Assuming that* $prec(p_1) = 1$*, the summary unearths the fact that all burglaries in private places in the community* south shore *led to arrests.*

4. Approximate Summarization

In Sec. 3, we have discussed how to generate summarized explanations for why and why-not questions. As mentioned in the introduction, we propose to use sampling to reduce the number of rule derivations we have to consider. Naturally, we would like to sample as early as possible in the process to avoid generating large intermediate results, e.g., the large set of failed derivations of a query result. Assume that a target sample size S is given as an input by the user (or automatically determined). The goal is to generate a sample of provenance and non-provenance that 1) contains representative patterns and 2) allows us to compute a close approximations of the real precision and recall of patterns. If it would be feasible to materialize the full set of derivations $\mathcal{A}(Q(\bar{c}), r)$ of a rule r, then randomly picking S derivations from this set would likely yield a representative sample. However, since we cannot materialize this set, picking a representative sample becomes more challenging. We would like the ratio between provenance and non-provenance in the sample to be similar to that ratio in $\mathcal{A}(Q(\bar{c}), r)$. The total number of rule derivations $|\mathcal{A}(Q(\bar{c}), r)|$ only depends on the size of the domains assigned by the user to attributes. For instance, assume the user has assigned the active domain $adom(D)$ as the domain for every attribute in the crime database and $|adom(D)| = 10,000$. Then, for a given why question, two of the 5 variables of rule r_0 (Fig. 1) are bound by the question. Thus, there are $10,000^3$ rule derivations. As evident from this example, if we know the size of the active domain, then the number of derivations in $\mathcal{A}(Q(\bar{c}), r)$ can be computed as $|adom(D)|$ to the power of the number of existential variables in rule r. We leave a formal analysis of approximation guarantees and sampling provenance for future work (e.g., in the spirit of [LWYZ16]).

Why questions. We now first explain the remaining steps of our sample approach. For why questions, we can compute the ratio as follows: 1) we compute the provenance for the user question since it is of moderate size and determine its size (denoted as $\#Prov$). Let $\#NProv$ denote the size of the non-provenance. We know that $|\mathcal{A}(Q(\bar{c}), r)| = \#Prov + \#NProv$ and thus $\#NProv = |\mathcal{A}(Q(\bar{c}), r)| - \#Prov$. We get

$$\frac{\#Prov}{\#NProv} = \frac{\#Prov}{|\mathcal{A}(Q(\bar{c}), r)| - \#Prov}$$

Given a target sample size S, this ratio allows us to determine how much provenance and non-provenance we should sample. For non-provenance, we still face the challenge that we want to create a sample of failed rule derivations without materializing the set of all failed rule derivations. Here, we can benefit from the fact that non-provenance for a why question is typically many orders of magnitude larger than provenance. Thus, derivations of a rule r sampled from the n-way crossproduct of the active domain where n is the number of existential variables in r will with high probability contain mostly non-provenance. Based on this observation, we devise the following method to create a sample of S tuples from the non-provenance: 1) we create n samples of size $S + \delta$ and zip them into a single sample with $S + \delta$ tuples with n attributes; 2) we check for each tuple whether it does correspond to a successful rule binding. If yes, then we remove this tuple from the sample; 3) if the resulting sample contains more than S tuples, then we randomly delete tuples until the desired sample size is reached. If the sample contains less than S tuples, then to create additional tuples we repeat steps 1-3. Here, δ should be chosen large enough that it is unlikely that we have to repeat steps 1-3. Note that this process can be implemented as SQL queries which produce the inputs to provenance graph generation and summarization. In fact, this is how we have implemented sampling in the PUG system.

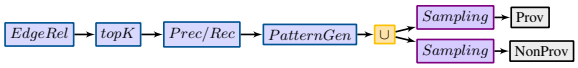

Figure 5: Overview of the summarization process

Why-not questions. The process of sampling for why-not questions is analog. The main difference is that materializing all rule derivations that belong to the provenance is not feasible. Thus, we use sampling to estimate the ratio $\frac{\#Prov}{\#NProv}$.

5. Integrating Capture with Summarization

In this section, we explain how to integrate summarization techniques with PUG's provenance capture mechanism that we have introduced in prior work [LKLG17]. Summarization and sampling are fully integrated into PUG's rewrite-based provenance capture mechanism. Given a request to compute and summarize an explanation for a user question, we first construct a rewritten program that captures provenance and generates the edge relation of the provenance graph as described in [LKLG17]. We, then, add additional instrumentation to implement sampling, generate patterns and evaluate their precision and recall, and select the top-k patterns. The result of this instrumentation is fed into the rules created by PUG to generate the edge relation of the provenance graph. Fig. 5 gives a high-level overview of the structure of the generated query. We, then, compile this query into SQL, evaluate it, and create a visualization of the summarized provenance graph.

6. Experiments

We have conducted a series of experiments to evaluate 1) the performance of combining sample-based summarization with provenance capture and 2) how sampling early in the process affects the quality of the produced summaries. We compare our sample-based approach (referred to as **S**ample **S**ummarization from now on) with an approach that creates summaries over the full provenance and non-provenance (called **F**ull from here on). Naturally, the **F** approach can only be applied to small instances because of the large number of rule derivations that it generates.

Setup. We use subsets of the Chicago crime dataset introduced in Example 1 where `Cx` denotes a subset with x rows. For the performance experiment (Fig. 6), we compute summaries for the query $\texttt{COMMCRIME}(C, T) :- \texttt{CRIMES}(I, T, L, C), \neg\texttt{ARREST}(I)$ which computes types of crimes that have not led to arrests in a particular community. Provenance questions why(new city,assault) and whynot(south shore,assault) are used in this experiment. We allocated a maximum of 2-hours for each summary computation. Computations that did not finish within 2 hours are omitted from graphs. To evaluate pattern quality and the amount of errors in precision and recall that is introduced by sampling (Fig. 7), we use the following query that returns types of crimes which did not lead to an arrest: $\texttt{EXISTCRIME}(T) :- \texttt{CRIMES}(I, T, L, C), \neg\texttt{ARREST}(I)$. All experiments were executed on a machine with 2 x 3.3Ghz AMD Opteron 4238 CPUs (12 cores in total) and 128GB RAM running Oracle Linux 6.4. We use the commercial DBMS X (name omitted due to licensing restrictions) as a backend.

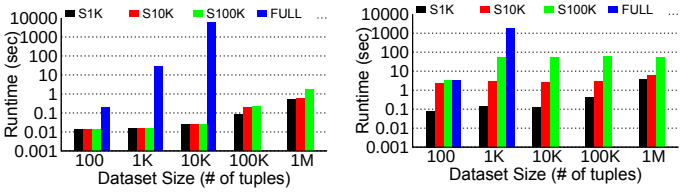

Figure 6: Runtime for generating summarized explanations for why (left) and why-not (right). Comparing sampling (e.g., S1K samples 1000 tuples) vs. capture and summarization without sampling.

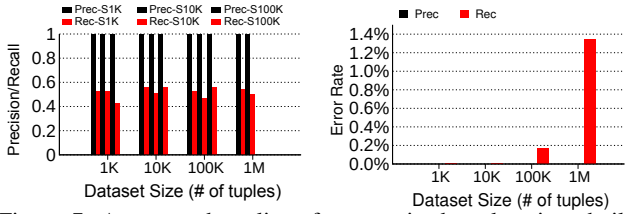

Figure 7: Aggregated quality of summarized explanations build from the top-3 patterns varying sample and database size (left); error in precision and recall introduced by sampling (right).

Results. The results of these experiments are shown in Fig. 6 and 7. First, we compare the performance of why and why-not summarization varying the sample and dataset size. As Fig. 6 shows, **SS** improves performance by several orders of magnitude compared to **F** with increasing benefit for larger database instances. By using sampling, we can capture and summarize the provenance for why and why-not questions within seconds whereas without sampling (**F**) runtimes are several minutes even for very small instances. To measure quality (Fig. 7), we consider the set of top-3 ranked patterns as an explanation, e.g., on possible such explanations may be (assault,I,Location,C) where Location has (street, apartment, sidewalk). The top-3 patterns are stable for a wide range of sample sizes (left in Fig. 7). The generated patterns have typically high precision and represent more than half of the provenance (the 100K sample over 1 million dataset did not finish with 2 hours and, thus, is omitted). The error of precision and recall introduced by sampling is mostly very low as shown on the right in Fig. 7 - the error rate is always 0 for precision and very small for recall.

7. Conclusions

We present an approach for computing summarized representations of provenance graphs for why and why-not questions. Our main goal is to generate summaries that are compact and descriptive as well as can be computed efficiently. To achieve this goal, we extend our previous work [LKLG17] for capturing provenance to explain (missing) answers first-order queries expressed in Datalog with negation. Our PUG [LKLG17] system instruments the input query Q to return the edge relation of the provenance graph explaining the (missing) answer of interest. In this work, we introduce additional instrumentation for generating patterns as candidate summaries and for evaluating the quality of these patterns to return only the top-k summaries. To make this feasible, we apply sampling early-on during provenance capture to reduce the cost of provenance capture and summarization. This is particularly important for explaining missing answers where the provenance is typically huge. In future work, we will investigate probabilistic worst-case bounds for the error in pattern quality introduced by our sampling method. Furthermore, we will investigate how integrity constraints can be exploited by our method to 1) reduce the search space size (e.g., explanations should not violate integrity contraints) and 2) for explaining a (missing) answer, i.e., the existence of a constraint may justify an answer [GKRL15].

References

[ABML09] M. K. Anand, S. Bowers, T. McPhillips, and B. Ludäscher. Efficient Provenance Storage over Nested Data Collections. In *EDBT*, pp. 958–969, 2009.

[ADDM14] E. Ainy, S. B. Davidson, D. Deutch, and T. Milo. Approximated Provenance for Complex Applications. In *TaPP*, 2014.

[CJR08] A. Chapman, H. V. Jagadish, and P. Ramanan. Efficient Provenance Storage. In *SIGMOD*, pp. 993–1006, 2008.

[DGM15] D. Deutch, A. Gilad, and Y. Moskovitch. Selective Provenance for Datalog Programs Using Top-K Queries. *PVLDB*, 8(12):1394–1405, 2015.

[EGAG+14] K. El Gebaly, P. Agrawal, L. Golab, F. Korn, and D. Srivastava. Interpretable and informative explanations of outcomes. *PVLDB*, 8(1):61–72, 2014.

[GKRL15] B. Glavic, S. Köhler, S. Riddle, and B. Ludäscher. Towards Constraint-based Explanations for Answers and Non-Answers. In *TaPP*, 2015.

[GKT07] T. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, pp. 31–40, 2007.

[HH10] M. Herschel and M. Hernandez. Explaining Missing Answers to SPJUA Queries. *PVLDB*, 3(1):185–196, 2010.

[ILJ84] T. Imieliński and W. Lipski Jr. Incomplete Information in Relational Databases. *JACM*, 31(4):761–791, 1984.

[KLP00] G. Kuper, L. Libkin, and J. Paredaens. *Constraint databases*. Springer Science & Business Media, 2000.

[KLZ13] S. Köhler, B. Ludäscher, and D. Zinn. First-Order Provenance Games. In *In Search of Elegance in the Theory and Practice of Computation*, pp. 382–399. Springer, 2013.

[LKLG17] S. Lee, S. Köhler, B. Ludäscher, and B. Glavic. A SQL-Middleware Unifying Why and Why-Not Provenance for First-Order Queries. In *ICDE*, pp. 485–496, 2017.

[LWYZ16] F. Li, B. Wu, K. Yi, and Z. Zhao. Wander Join: Online Aggregation via Random Walks. In *SIGMOD*, pp. 615–629, 2016.

[MGMS10] A. Meliou, W. Gatterbauer, K. Moore, and D. Suciu. The Complexity of Causality and Responsibility for Query Answers and non-Answers. *PVLDB*, 4(1):34–45, 2010.

[OZ11] D. Olteanu and J. Závodný. On Factorisation of Provenance Polynomials. In *TaPP*, 2011.

[RKL14] S. Riddle, S. Köhler, and B. Ludäscher. Towards Constraint Provenance Games. In *TaPP*, 2014.

[RS14] S. Roy and D. Suciu. A formal approach to finding explanations for database queries. In *SIGMOD*, pp. 1579–1590, 2014.

[tCCST15] B. ten Cate, C. Civili, E. Sherkhonov, and W.-C. Tan. High-level why-not explanations using ontologies. In *PODS*, pp. 31–43, 2015.

[WDM15] X. Wang, X. L. Dong, and A. Meliou. Data X-Ray: A Diagnostic Tool for Data Errors. In *SIGMOD*, pp. 1231–1245, 2015.

[WM13] E. Wu and S. Madden. Scorpion: Explaining Away Outliers in Aggregate Queries. *PVLDB*, 6(8):553–564, 2013.

[XE14] D. Xiao and M. Y. Eltabakh. InsightNotes: Summary-based annotation management in relational databases. In *SIGMOD*, pp. 661–672, 2014.