Short and Simple Explanations for Datalog Derivations

Jelle Hellings and Yuchen You

McMaster University, 1280 Main St. W., Hamilton, ON L8S 4L7, Canada
{jhellings, youy10}@mcmaster.ca

Abstract. Simple and short query explanations that detail how a query result is obtained from an input dataset are a powerful way to gain insight into the results of queries. Unfortunately, simple and short explanations for recursive queries are non-trivial: query results of recursive queries typically have a huge set of equally-valid explanations. Toward providing simple and short query explanations, we believe that there is a strong need for an improved understanding of *explanations* of recursive queries. In this paper, we present such a framework by studying explanations of facts derived using Datalog derivation rules. To enable *simple* explanations, we present a standard tree-based formal representation of explanations and we formalize four *conceptually distinct* types of simple explanations based on their tree structure. Next, we show how to *effectively compute* two types of simple explanations, which opens the door for practical algorithms to explain Datalog query results. Finally, we prove that computing the other two types of simple explanations involves solving NP-complete decision problems and, hence, is inherently hard.

Keywords: Datalog · Query Explanation · Evaluation Algorithm

1 Introduction

Understanding why a given tuple is in the output of a simple join-select-project query (e.g., elementary SQL queries) is typically straightforward: one can easily trace the tuples in the output to corresponding tuples in the input tables. Unfortunately, this simplicity disappears quickly when trying to explain the results of more complex queries.

As an important example of complex queries, we can consider queries whose definition and evaluation involves *recursion*. Such recursive queries are central in relational and graph querying and have applications in data integration, automated reasoning, AI, computational linguistics, information retrieval, logic programming, and descriptive logic [2, 3, 5, 10, 12, 16, 18, 23, 24, 29]. Due to the presence of recursion in queries, individual values in the output of recursive query results can have a *large* or even *unbounded* number of valid distinct explanations [7, 24]. Hence, understanding or explaining query results of recursive queries remains a challenge.

In this paper, we make the next step toward formal *explanations* of recursive queries by studying the explanation of queries expressed in Datalog [1, 20], a

powerful logic-based recursive query formalism for relational and graph data. One can *explain* a derived Datalog fact via the individual derivation steps used to derive the fact. Typically, these derivation steps can be represented via a tree-based format. To obtain such *tree-based explanations*, one can use variants of standard Datalog evaluation algorithms that keep track of the derivation steps made during evaluation. The explanations obtained in this way are not guaranteed to be easy to understand, however. Hence, for *practical* explanations, we need a way to find *short and simple* explanations.

We note that obtaining explanation is closely related to the general question of *why*-provenance for Datalog [6–8, 15, 17, 30], the main difference being that *why*-provenance focuses on explaining from which parts of the underlying data a fact can be derived, this *without providing an explicit explanation* of the derivation steps that can derive the fact.

Toward *short and simple* Datalog explanations, we formalize simple explanations and study the complexity of computing these explanations. In specific:

1. First, in Section 3, we show how we will use *derivation trees* as the formal tree-based representation of Datalog derivations (and, hence, explanations).
2. Second, in Section 4, we formalize four complexity metrics based on the *structure* of these derivation trees: base fact complexity, derivation complexity, unique derivation complexity, and derivation depth complexity.
3. Third, in Theorem 1, we prove that these four types of complexity metrics are distinct: an explanation that minimizes one of these metrics does not necessarily minimize the other three. Hence, this gives rise to four distinct types of *simple* explanations
4. Then, in Section 5, we show that simple explanations based on *derivation complexity* and *derivation depth complexity* can be practically computed by annotating standard Datalog evaluation algorithms. This opens the door for practical derivation engines to integrate explanations in their evaluation algorithms.
5. Next, in Section 6, we show that simple explanations based on *base fact complexity* and *unique derivation complexity* cannot be practically computed under the common assumption that $\mathsf{P} \neq \mathsf{NP}$: the *data complexity* of the decision problem determining whether a simple explanation exists under these complexity measures is NP-complete.
6. Finally, in Section 7, we discuss the practicality of simple explanations.

Furthermore, in Section 2, we introduce the notation and terminology used in this paper; in Section 8, we discuss the related work; and in Section 9 we conclude on our findings.

2 Preliminaries

We assume a universe of constants $\mathbb{U}$. In this paper, we study Datalog programs [1, 20] with recursion, with inequalities, but without negation.

Definition 1. *A Datalog program is a pair $\langle B, D\rangle$ in which B is a set of Datalog rules defining* base facts *and D is a set of Datalog rules defining* derived facts. *An n-ary Datalog rule in B (a base rule) is of the form "head $\leftarrow \bullet$" in which the* head *is of the form $R(\mathcal{U})$ with R the name of the rule and $\mathcal{U}$ a list of n constants. An n-ary Datalog rule in D (a derivation rule) is of the form "head $\leftarrow$ body" in which*

1. *the* head *is of the form $R(\mathcal{V})$ with R the name of the rule and $\mathcal{V}$ a list of n free variables of the rule; and*
2. *the* body *consists of a non-empty conjunction of terms of the form $S(\mathcal{W})$, with S the name of an m-ary Datalog rule and $\mathcal{W}$ a list of m variables, or of the form $V \neq W$ with V, W variables.*

The arity *of a rule with a head of the form $R(\mathcal{V})$, with $\mathcal{V}$ a list of n free variables, is $|\mathcal{V}| = n$. We require that the body of a derivation rule $R(\mathcal{V})$ must use all variables in $\mathcal{V}$.*

To simplify presentation, we assume that all Datalog rules in a program $\langle B, D\rangle$ for a given rule name R have the same arity and are of the same type (either all rules are base rules or all rules are derivation rules).

Example 1. Consider the graph of Figure 1, *left*. This graph represents the following base facts:

$$\begin{aligned}
\mathsf{ParentOf}(\texttt{Alicia}, \texttt{Dafni}) &\leftarrow \bullet, & \mathsf{ParentOf}(\texttt{Alicia}, \texttt{Ivan}) &\leftarrow \bullet,\\
\mathsf{ParentOf}(\texttt{Bo}, \texttt{Eva}) &\leftarrow \bullet, & \mathsf{ParentOf}(\texttt{Celeste}, \texttt{Dafni}) &\leftarrow \bullet,\\
\mathsf{ParentOf}(\texttt{Dafni}, \texttt{Eva}) &\leftarrow \bullet, & \mathsf{ParentOf}(\texttt{Gale}, \texttt{Frieda}) &\leftarrow \bullet,\\
\mathsf{ParentOf}(\texttt{Eva}, \texttt{Harry}) &\leftarrow \bullet;
\end{aligned}$$

$$\begin{aligned}
\mathsf{FriendOf}(\texttt{Celeste}, \texttt{Gale}) &\leftarrow \bullet, & \mathsf{FriendOf}(\texttt{Gale}, \texttt{Celeste}) &\leftarrow \bullet,\\
\mathsf{FriendOf}(\texttt{Frieda}, \texttt{Eva}) &\leftarrow \bullet, & \mathsf{FriendOf}(\texttt{Eva}, \texttt{Frieda}) &\leftarrow \bullet.
\end{aligned}$$

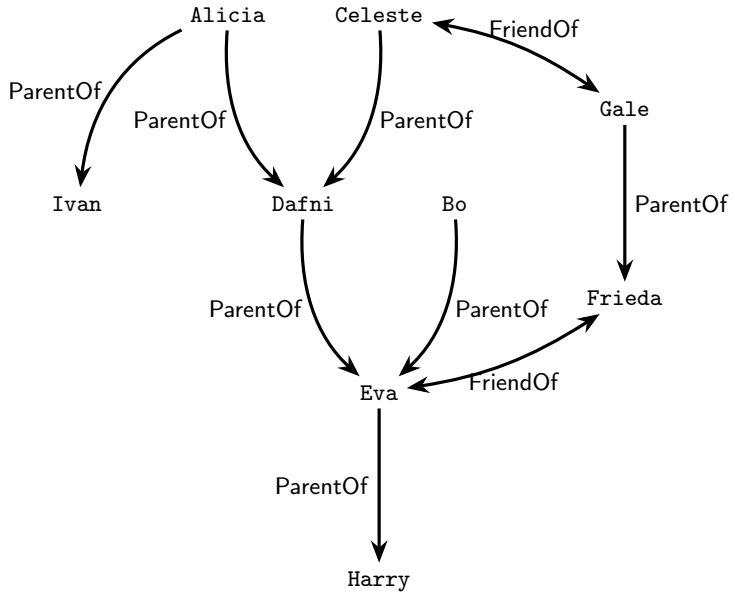
This dataset conveys many additional relationships between people that can be derived using derivation rules. Consider the 2-ary derivation rules

$$\begin{aligned}
\mathsf{GrandparentOf}(v, w) &\leftarrow \mathsf{ParentOf}(v, u) \wedge \mathsf{ParentOf}(u, w);\\
\mathsf{SiblingOf}(v, w) &\leftarrow \mathsf{ParentOf}(u, v) \wedge \mathsf{ParentOf}(u, w) \wedge v \neq w.
\end{aligned}$$

On the above dataset, these Datalog rules evaluate to the derived facts in Figure 1, *right*. Datalog rules can utilize recursion to derive even more complex relationships. For example,

$$\begin{aligned}
\mathsf{AncestorOf}(v, w) &\leftarrow \mathsf{ParentOf}(v, w);\\
\mathsf{AncestorOf}(v, w) &\leftarrow \mathsf{ParentOf}(v, u) \wedge \mathsf{AncestorOf}(u, w).
\end{aligned}$$

On the above dataset, this Datalog rule evaluates to the derived facts that include all ParentOf pairs, all GrandparentOf pairs, and the pairs $(\texttt{Alicia}, \texttt{Harry})$ and $(\texttt{Celeste}, \texttt{Harry})$.



GrandparentOf	
Alicia	Eva
Celeste	Eva
Dafni	Harry
Bo	Harry

SiblingOf	
Dafni	Ivan
Ivan	Dafni

Fig. 1. A simple social network illustrating the relationships between people.

There are often many ways to derive the same facts. For example, the above AncestorOf rules derive the same relationship between people as the following alternative rules:

$$\mathsf{AltAncestorOf}(v, w) \leftarrow \mathsf{ParentOf}(v, w);$$
$$\mathsf{AltAncestorOf}(v, w) \leftarrow \mathsf{AltAncestorOf}(v, u) \wedge \mathsf{AltAncestorOf}(u, w).$$

The semantics for the above Datalog rules are self-explanatory. Next, we formalize the proof-theoretic semantics for Datalog programs that we use in this paper [1, 20].

Definition 2. *Let R be the name of an n-ary rule in Datalog program $\langle B, D\rangle$. The tuple $(c_1, \ldots, c_n)$ is in the evaluation of R if either*

1. *R is the name of a base rule and the rule "$R(c_1, \ldots, c_n) \leftarrow \bullet$" $\in B$ exists; or*
2. *R is the name of a derivation rule, the rule*

$$\text{“}R(v_1, \ldots, v_n) \leftarrow S_1(w_{1,1}, \ldots, w_{1,m_1}) \wedge \cdots \wedge S_j(w_{j,1}, \ldots, w_{j,m_j})\text{”} \in D$$

exists, and a variable assignment $\mu : \{w_{1,1}, \ldots, w_{j,m_j}\} \to \mathbb{U}$ exists such that $\mu(v_1) = c_1, \ldots, \mu(v_n) = c_n$ and, for all $i, 1 \leq i \leq j$, the tuple $(\mu(w_{i,1}), \ldots, \mu(w_{i,m_i}))$ is in the evaluation of rule S_i.

Example 2. Consider the example Datalog rules of Example 1 as visualized in Figure 1. The tuple (`Celeste`, `Harry`) is in the evaluation of rule AncestorOf because

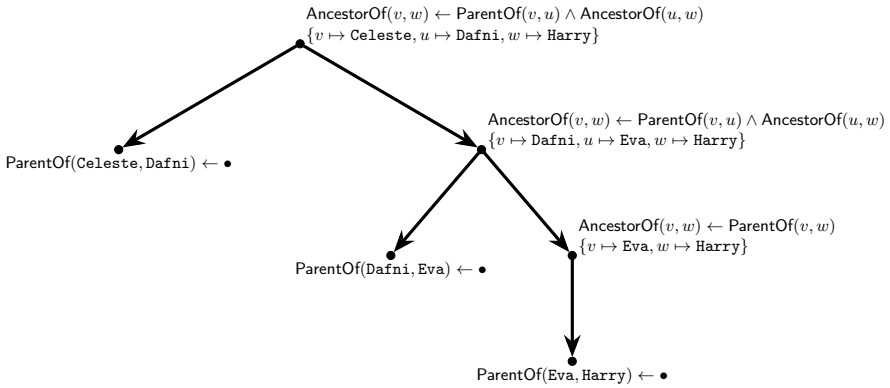


Fig. 2. The derivation tree of the fact AncestorOf(Celeste, Harry)

1. The tuples (Celeste, Dafni), (Dafni, Eva), and (Eva, Harry) are in the evaluation of ParentOf because the base rules ParentOf(Celeste, Dafni) ← •, ParentOf(Dafni, Eva) ← •, and ParentOf(Eva, Harry) ← • exist.
2. The tuple (Eva, Harry) is in the evaluation of AncestorOf because the derivation rule AncestorOf(v, w) ← ParentOf(v, w) and variable assignment $\{v \mapsto$ Eva$, w \mapsto$ Harry$\}$ exists such that (Eva, Harry) is in the evaluation of ParentOf.
3. The tuple (Dafni, Harry) is in the evaluation of AncestorOf because the derivation rule AncestorOf(v, w) ← ParentOf(v, u) ∧ AncestorOf(u, w) and variable assignment $\{v \mapsto$ Dafni$, u \mapsto$ Eva$, w \mapsto$ Harry$\}$ exists such that (Dafni, Eva) is in the evaluation of ParentOf and (Eva, Harry) is in the evaluation of AncestorOf.
4. Finally, the tuple (Celeste, Harry) is in the evaluation of AncestorOf because the derivation rule AncestorOf(v, w) ← ParentOf(v, u) ∧ AncestorOf(u, w) and variable assignment $\{v \mapsto$ Celeste$, u \mapsto$ Dafni$, w \mapsto$ Harry$\}$ exists such that (Celeste, Dafni) is in the evaluation of ParentOf and (Dafni, Harry) is in the evaluation of AncestorOf.

3 Explaining Datalog

Example 2 *explains* why a tuple is in the evaluation of a Datalog rule by providing a *step-by-step proof* of the derivation of the tuple. Such proofs are typically represented by a *derivation tree*. Next, we formalize these derivation trees, which we will then use to represent explanations and to reason about the complexity of explanations.

Definition 3. *Consider a derivation for the tuple* $(c_1, \ldots, c_n)$ *in the evaluation of the rule named* R*. The* derivation tree *for fact* $R(c_1, \ldots, c_n)$ *is defined recursively as follows:*

1. *if R is the name of a base rule and the rule $r =$ "$R(c_1, \ldots, c_n) \leftarrow \bullet$" $\in B$ exists, then this derivation step is represented by a* leaf node *labeled r; and*
2. *if R is the name of a derivation rule and the explanation of fact $R(c_1, \ldots, c_n)$ uses the rule*

$$r = \text{"}R(v_1, \ldots, v_n) \leftarrow S_1(w_{1,1}, \ldots, w_{1,m_1}) \wedge \cdots \wedge S_j(w_{j,1}, \ldots, w_{j,m_j})\text{"} \in D$$

and variable assignment $\mu : \{w_{1,1}, \ldots, w_{j,m_j}\} \to \mathbb{U}$, then this derivation step is represented by an internal node *labeled r, μ with, for all $i, 1 \leq i \leq j$, a child representing the derivation of the fact $S_i(\mu(w_{i,1}), \ldots, \mu(w_{i,m_i}))$.*

Example 3. Consider the derivation in Example 2 of fact $\mathsf{AncestorOf}(\texttt{Celeste}, \texttt{Harry})$. In Figure 2, we visualized the derivation tree for this fact. The derivation visualized in Figure 2 is the *only* possible derivation of fact $\mathsf{AncestorOf}(\texttt{Celeste}, \texttt{Harry})$. In many practical cases, such a unique derivation does not exist, however. Consider the rule name Edge representing the graph with edges $(n_1, n_2), (n_2, n_3), (n_3, n_1)$ (a cycle with three nodes) and consider the derivation rules

$$\mathsf{TC}(v, w) \leftarrow \mathsf{Edge}(v, w);$$
$$\mathsf{TC}(v, w) \leftarrow \mathsf{TC}(v, u) \wedge \mathsf{TC}(u, w).$$

In Figure 3, we visualized the derivation tree for the tuple (n_1, n_3) in the evaluation of TC. This explanation is not the simplest, however: the green-colored subtree in the left bottom is itself a much simpler derivation of the same fact $\mathsf{TC}(n_1, n_3)$.

Most Datalog evaluation algorithms can be annotated to not only determine all tuples in the evaluation of a rule name, but also to provide an explanation of each such tuple (by keeping track of the derivation steps made during evaluation). We refer to Figure 4 for an illustration on how a standard *semi-naive* evaluation algorithm for Datalog [1, 20, 21, 28] can be annotated for this purpose.

4 Short and Simple Datalog Explanations

As illustrated in Example 3 and Figure 3, not all explanations are equal. Furthermore, the derivation of $\mathsf{TC}(n_1, n_3)$ illustrated in Figure 3 can easily be bootstrapped into an infinite number of distinct explanations of $\mathsf{TC}(n_1, n_3)$. Hence, to maximize the *readability* and *understandability* of an explanation, we need to obtain *short and simple* explanations. Before we can reason about whether an explanation is short and simple, we first need to determine what the *complexity* of an explanation is. Next, we define four alternative *cost measures* for the complexity of an explanation based on structural properties on their derivation trees.

Definition 4. *Consider a derivation for the fact f in the evaluation of the Datalog rule named R and consider a derivation tree $\mathcal{T}$ explaining this fact.*

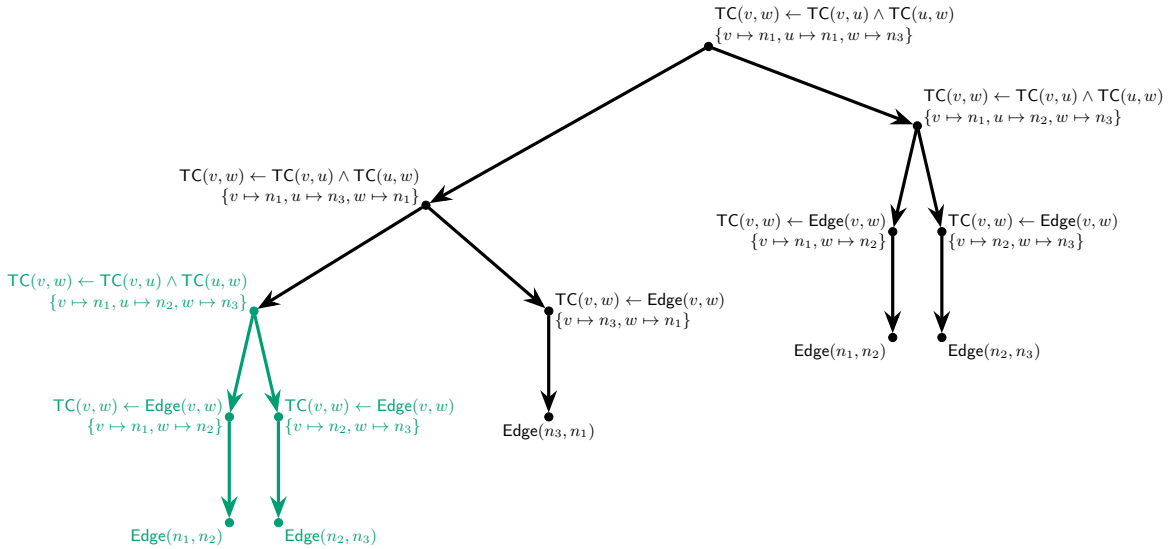


Fig. 3. The derivation tree of the tuple (n_1, n_3) in the evaluation of the rule named TC on the three-node cycle graph with nodes $\{n_1, n_2, n_3\}$. Note that the green-colored subtree in the left bottom is itself a simpler derivation of the same fact $\mathsf{TC}(n_1, n_3)$.

1. *The* base-fact complexity $\mathcal{C}$*-base*$(\mathcal{T})$ *is defined by the number of distinct leaf node labels used in* $\mathcal{T}$ *(hence, the number of distinct base rules used in the explanation).*
2. *The* derivation complexity $\mathcal{C}$*-deriv*$(\mathcal{T})$ *is defined by the number of internal nodes used in* $\mathcal{T}$ *(hence, the total number of derivation steps used in the explanation).*
3. *The* unique derivation complexity $\mathcal{C}$*-uderiv*$(\mathcal{T})$ *is defined by the number of distinct internal node labels used in* $\mathcal{T}$ *(hence, the total number of unique derivation steps used in the explanation, in which the repeated use of a derivation of a fact only counts once).*
4. *The* derivation depth complexity $\mathcal{C}$*-ddepth*$(\mathcal{T})$ *is defined by the length of the longest path in* $\mathcal{T}$ *from the root of the tree to a leaf (hence, the depth of the derivation steps used in the explanation).*

Example 4. Let $\mathcal{T}$ be the derivation tree visualized in Figure 3 and let $\mathcal{T}'$ be the green-colored subtree in the left bottom of $\mathcal{T}$. Both $\mathcal{T}$ and $\mathcal{T}'$ explain the derivation of fact $\mathsf{TC}(n_1, n_3)$. We have

$$\begin{aligned} \mathcal{C}\text{-base}(\mathcal{T}) &= 3, & \mathcal{C}\text{-base}(\mathcal{T}') &= 2; \\ \mathcal{C}\text{-deriv}(\mathcal{T}) &= 9, & \mathcal{C}\text{-deriv}(\mathcal{T}') &= 3; \\ \mathcal{C}\text{-uderiv}(\mathcal{T}) &= 6, & \mathcal{C}\text{-uderiv}(\mathcal{T}') &= 3; \\ \mathcal{C}\text{-ddepth}(\mathcal{T}) &= 4, & \mathcal{C}\text{-ddepth}(\mathcal{T}') &= 2. \end{aligned}$$

Algorithm SemiNaive-Annotated($\langle B, D \rangle$)**:**

We write leaf-node(r) to denote the leaf node with label r.
result := a dictionary with $\{h \mapsto \text{leaf-node}(\text{“}h \leftarrow \bullet\text{”}) \mid \text{“}h \leftarrow \bullet\text{”} \in B\}$.
new := a queue initialized with facts $\{h \mid \text{“}h \leftarrow \bullet\text{”} \in B\}$.
done := $\emptyset$.
while *new* $\neq \emptyset$ **do**
  Remove fact $S(d_1, \ldots, d_n)$ from the bottom of *new* and add it to *done*
  while there exists a derivation rule

$$\text{r} = \text{“}R(v_1, \ldots, v_m) \leftarrow S_1(w_{1,1}, \ldots, w_{1,m_1}) \wedge \cdots \wedge S_j(w_{j,1}, \ldots, w_{j,m_j})\text{”} \in D$$

and variable assignment $\mu : \{w_{1,1}, \ldots, w_{j,m_j}\} \to \mathbb{U}$ such that

i. there exists an i with $S(d_1, \ldots, d_n) = S_i(\mu(w_{i,1}), \ldots, \mu(w_{i,m_i}))$, $1 \leq i \leq j$;
ii. for each $i, 1 \leq i \leq j$, we have $S_i(\mu(w_{i,1}), \ldots, \mu(w_{i,m_i})) \in$ *done*; and
iii. $R(\mu(v_1), \ldots, \mu(v_m)) \notin (\textit{done} \cup \textit{new})$

  do
    Add $R(\mu(v_1), \ldots, \mu(v_m))$ to the top of *new*.
    $n :=$ an internal node with label r, μ and j children *result*$(S_i(\mu(w_{i,1}), \ldots, \mu(w_{i,m_i})))$, $1 \leq i \leq j$.
    Add $R(\mu(v_1), \ldots, \mu(v_m)) \mapsto n$ to *result*.
  end while
end while
return *result*.

Fig. 4. A variant of the standard semi-naive Datalog evaluation algorithm that is annotated toward constructing derivation trees for each fact. In this algorithm, the colored parts indicate the annotations that construct derivation trees.

According to all four complexity measures we introduced, the tree $\mathcal{T}'$ has lower complexity than tree $\mathcal{T}$. Hence, the explanation provided by tree $\mathcal{T}'$ is simpler than the one provided by $\mathcal{T}$. Furthermore, the derivation tree $\mathcal{T}'$ is the smallest-possible tree of the fact $\mathsf{TC}(n_1, n_3)$. Hence, we are justified to say that the derivation tree $\mathcal{T}'$ provides a *simple* explanation.

Example 4 illustrates an explanation of a derived fact that minimizes all four cost metrics we introduced. Furthermore, this explanation is *unique*: all other possible explanations have higher costs. Unfortunately, such a unique explanation that minimizes all four cost metrics is not guaranteed to exist, as we shall prove next:

Theorem 1. *Let $\mathcal{C}_1, \mathcal{C}_2 \in \{\mathcal{C}\text{-base}, \mathcal{C}\text{-deriv}, \mathcal{C}\text{-uderiv}, \mathcal{C}\text{-ddepth}\}$, $\mathcal{C}_1 \neq \mathcal{C}_2$, be two distinct cost measures. There exist a Datalog program with rule name R and with tuple t in the evaluation of R such that any derivation of fact $R(t)$ with minimal $\mathcal{C}_1$ complexity does not have minimal $\mathcal{C}_2$ complexity.*

Proof. We can prove each of the twelve cases $\mathcal{C}_1, \mathcal{C}_2 \in \{\mathcal{C}\text{-base}, \mathcal{C}\text{-deriv}, \mathcal{C}\text{-uderiv}, \mathcal{C}\text{-ddepth}\}$, $\mathcal{C}_1 \neq \mathcal{C}_2$ by providing an explanation that is minimal with respect to $\mathcal{C}_1$, but not with respect to $\mathcal{C}_2$. In each case, we can use base facts that we specify in the form of the edge-labeled graph of Figure 5.

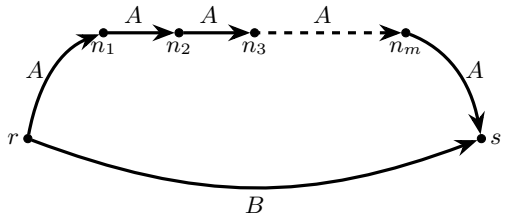


Fig. 5. A graph with two paths connecting nodes r and s: one A-labeled path of $m+1$ edges and one B-labeled path of one edge.

This graph represents $m+2$ base facts: $m+1$ edges labeled A, and one edge labeled B. In each case, we use the following derivation rules:

$$\begin{aligned}&\mathsf{Path}\text{-}A(v,w) \leftarrow A(v,w),\\ &\mathsf{Path}\text{-}A(v,w) \leftarrow A(v,u) \wedge \mathsf{Path}\text{-}A(u,w)\end{aligned}$$

There is only one way to explain $\mathsf{Path}\text{-}A(r,s)$: this explanation uses all $m+1$ edges labeled A and uses $m+1$ internal nodes. Next, we illustrate the proof for the case $\mathcal{C}_1 = \mathcal{C}$-base. All other cases are analogous.

Let $\mathcal{C}_1 = \mathcal{C}$**-base and** $\mathcal{C}_2 \in \{\mathcal{C}$**-deriv**, $\mathcal{C}$**-uderiv**, $\mathcal{C}$**-ddepth**$\}$. Consider the following derivation rules:

$$\begin{aligned}&\mathsf{Path}\text{-}A'(v,w) \leftarrow A(v,w),\\ &\mathsf{Path}\text{-}A'(v,w) \leftarrow A(v,u) \wedge \mathsf{Path}\text{-}A'(u,w);\\ &\mathsf{Path}\text{-}A''(v,w) \leftarrow \mathsf{Path}\text{-}A'(u,w);\\ &\mathsf{Ans}(v,w) \leftarrow \mathsf{Path}\text{-}A(v,w) \wedge \mathsf{Path}\text{-}A''(v,w),\\ &\mathsf{Ans}(v,w) \leftarrow \mathsf{Path}\text{-}A(v,w) \wedge B(v,w).\end{aligned}$$

There are two possible explanations for the fact $\mathsf{Ans}(r,s)$: one that uses the rule $\mathsf{Ans}(v,w) \leftarrow \mathsf{Path}\text{-}A(v,w) \wedge \mathsf{Path}\text{-}A'(v,w)$, and another that uses the rule $\mathsf{Ans}(v,w) \leftarrow \mathsf{Path}\text{-}A(v,w) \wedge \mathsf{Path}\text{-}B(v,w)$. Let $\mathcal{T}$ and $\mathcal{T}'$ be the derivation trees for the first and second explanation, respectively.

The tree $\mathcal{T}$ uses only all $m+1$ A-labeled edges as base facts, $m+1$ internal nodes for explaining $\mathsf{Path}\text{-}A(r,s)$, $m+1$ internal nodes for explaining $\mathsf{Path}\text{-}A'(r,s)$, one additional internal node for explaining $\mathsf{Path}\text{-}A''(r,s)$, and one additional internal node for explaining $\mathsf{Ans}(r,s)$. The longest path from the root of $\mathcal{T}$ to a leaf is $m+3$: the path goes via the internal node explaining $\mathsf{Path}\text{-}A''(r,s)$ to the

node explaining Path-$A'(r,s)$, and the depth of the tree rooted at Path-$A'(r,s)$ is $m+1$.

The tree $\mathcal{T}'$ uses all $m+2$ edges as base facts, $m+1$ internal nodes for explaining Path-$A(r,s)$, and one additional internal node for explaining $\mathsf{Ans}(r,s)$. The longest path from the root of $\mathcal{T}'$ to a leaf is $m+2$: the path goes from the root to the node explaining Path-$A(r,s)$, and the depth of the tree rooted at Path-$A(r,s)$ is $m+1$.

We conclude

$$\begin{aligned} \mathcal{C}\text{-base}(\mathcal{T}) &= m+1, & \mathcal{C}\text{-base}(\mathcal{T}') &= m+2; \\ \mathcal{C}\text{-deriv}(\mathcal{T}) &= 2(m+1)+2, & \mathcal{C}\text{-deriv}(\mathcal{T}') &= m+2; \\ \mathcal{C}\text{-uderiv}(\mathcal{T}) &= 2(m+1)+2, & \mathcal{C}\text{-uderiv}(\mathcal{T}') &= m+2; \\ \mathcal{C}\text{-ddepth}(\mathcal{T}) &= m+3, & \mathcal{C}\text{-ddepth}(\mathcal{T}') &= m+2. \end{aligned}$$

Hence, $\mathcal{T}$ is the only explanation with minimal $\mathcal{C}$-base complexity and does not have minimal $\mathcal{C}$-deriv, $\mathcal{C}$-uderiv, and $\mathcal{C}$-ddepth complexity.

Due to Theorem 1, it is justifiable to consider explanations with low complexity according to only one of the four cost measurements we introduced.

Definition 5. *Consider a derivation for the tuple t in the evaluation of the Datalog rule named R and consider a derivation tree $\mathcal{T}$ explaining fact $R(t)$. We say that tree $\mathcal{T}$ is* $\mathcal{C}$-simple, $\mathcal{C} \in \{\mathcal{C}\text{-}base, \mathcal{C}\text{-}deriv, \mathcal{C}\text{-}uderiv, \mathcal{C}\text{-}ddepth\}$ *if, for every derivation tree $\mathcal{T}'$ explaining fact $R(t)$, we have $\mathcal{C}(\mathcal{T}) \leq \mathcal{C}(\mathcal{T}')$.*

5 Computing Simple Explanations

We have established *four* types of simple explanations based on four distinct complexity measures for the derivation trees representing explanations. Next, we will provide algorithms to compute $\mathcal{C}$-deriv-simple and $\mathcal{C}$-ddepth-simple explanations.

First, we consider the computation of $\mathcal{C}$-deriv-simple explanations. For this task, we introduce the SNA-PQ algorithm of Figure 6. This algorithm derives new facts in a similar way as the standard semi-naive Datalog evaluation algorithm [1, 20, 21, 28], except that SNA-PQ will explore derivation steps involving newly derived facts in *derivation complexity* order. To do so, the algorithm uses a priority queue to store newly derived facts ordered on their derivation complexity cost and process them based on this order, this in a similar way as the usage of a priority queue to process nodes based on their distance to a source node in Dijkstra's algorithm [11, Section 22.3]. We note that the SNA-PQ algorithm is a generalization of the MinimizeSetGG [24] algorithm from context-free path queries to general Datalog rules and is similar to the *Dijkstra-like algorithm* of Alrabbaa et al. [2] that can be used to find optimal proofs for entailment in descriptive logic.

Theorem 2. *Let R be the name of an n-ary rule in Datalog program $\langle B, D\rangle$. The pair $R(c_1, \ldots, c_n) \mapsto z$ is in the result of SNA-PQ($\langle B, D\rangle$) if and only*

Algorithm SNA-PQ($\langle B, D \rangle$)**:**

We write leaf-node(r) to denote the leaf node with label r.
result := a dictionary with $\{h \mapsto \text{leaf-node}(\text{“}h \leftarrow \bullet\text{”}) \mid \text{“}h \leftarrow \bullet\text{”} \in B\}$.
new := a priority queue initialized with facts $\{h \mid \text{“}h \leftarrow \bullet\text{”} \in B\}$.
/* This priority queue is ordered on the *derivation complexity* of facts. */
done := $\emptyset$.
while *new* $\neq \emptyset$ **do**
  Remove fact $S(d_1, \ldots, d_n)$ with lowest priority from *new* and add it to *done*.
  while there exists a derivation rule

$$\text{r} = \text{“}R(v_1, \ldots, v_m) \leftarrow S_1(w_{1,1}, \ldots, w_{1,m_1}) \wedge \cdots \wedge S_j(w_{j,1}, \ldots, w_{j,m_j})\text{”} \in D$$

and variable assignment $\mu : \{w_{1,1}, \ldots, w_{j,m_j}\} \to \mathbb{U}$ such that

i. there exists an i with $S(d_1, \ldots, d_n) = S_i(\mu(w_{i,1}), \ldots, \mu(w_{i,m_i}))$, $1 \leq i \leq j$;
ii. for each $i, 1 \leq i \leq j$, we have $S_i(\mu(w_{i,1}), \ldots, \mu(w_{i,m_i})) \in$ *done*; and
iii. $R(\mu(v_1), \ldots, \mu(v_m)) \notin$ *done*

  do
    if $R(\mu(v_1), \ldots, \mu(v_m))$ is already in *new* **then**
      Compare this new derivation for $R(\mu(v_1), \ldots, \mu(v_m))$ with the one already stored in *result* and determine which derivation has the lowest *derivation complexity*. Update the priority for $R(\mu(v_1), \ldots, \mu(v_m))$ in *new* and update the derivation in *result* to represent the derivation with the lowest *derivation complexity*.
    else
      Add $R(\mu(v_1), \ldots, \mu(v_m))$ to the top of *new*.
      n := an internal node with label r, μ and j children *result*$(S_i(\mu(w_{i,1}), \ldots, \mu(w_{1,m_i})))$, $1 \leq i \leq j$.
      Add $R(\mu(v_1), \ldots, \mu(v_m)) \mapsto n$ to *result*.
    end if
  end while
end while
return *result*.

Fig. 6. A variant of the standard semi-naive Datalog evaluation algorithm that is annotated toward constructing $\mathcal{C}$-deriv-simple derivation trees for each fact. To ensure derivation trees are constructed with *increasing* derivation complexity, the order in which facts are constructed is managed by the priority queue *new* (that holds each newly-created fact ordered on the derivation complexity of their derivation).

if tuple $(c_1, \ldots, c_n)$ *is in the evaluation of* R*. Furthermore,* z *is the root of a derivation tree that is a* $\mathcal{C}$*-deriv-simple explanation of fact* $R(c_1, \ldots, c_n)$*.*

Proof. To simplify notation, we use the shorthand $\mathsf{cost}(f)$ to denote the $\mathcal{C}$-deriv-complexity of the derivation in *result* for fact f and we use the shorthand

$\mathsf{optimal}(f)$ to denote the $\mathcal{C}$-deriv-complexity of any $\mathcal{C}$-deriv-simple explanation of fact f. We assume $\mathsf{cost}(f) = \infty$ for facts $f \notin \mathit{result}$.

Let x be the cost of the first fact in new, starting at $x = 0$ when initially entering the loop at Line 5. One can prove this proposition by proving that the following invariants hold on the loop at Line 5:

(i) The facts $g \in \mathit{new}$ are ordered on increasing $\mathsf{cost}(g)$.
(ii) For all facts g, we either have:
  (a) fact g is in done with $\mathsf{cost}(g) = \mathsf{optimal}(g) \leq x$; or
  (b) fact g is in new with $x \leq \mathsf{cost}(g) = \mathsf{optimal}(g)$; or
  (c) $\mathsf{cost}(g) > \mathsf{optimal}(g) > x$ and, for every tree $\mathcal{T}$ representing a $\mathcal{C}$-deriv-simple explanation of fact g and every fact $g' \in \mathcal{T}$ with $\mathsf{cost}(g') > \mathsf{optimal}(g')$, every path from g' to a leaf in $\mathcal{T}$ will reach a fact g'' such that $g'' \in (\mathit{new} \cup \mathit{done})$ and $\mathsf{cost}(g'') = \mathsf{optimal}(g'')$, and at least one of these reachable facts g'' is in new.

After Lines 2–4, result and new hold all base facts, no facts are in done, and all facts $g \in \mathit{new}$ are base facts with $\mathsf{cost}(g) = \mathsf{optimal}(g) = 0$. As all derived facts g have $\mathsf{optimal}(g) \geq 1$, no derived facts are included in $\mathit{done} \cup \mathit{new}$, and all explanations end in base facts, the Invariants (i)–(ii) initially hold.

Now assume the Invariants (i)–(ii) hold for the first i iterations of the loop at Line 5. Next, we consider the $i + 1$-th iteration of the loop at Line 5. As new is a priority queue (Line 3), Invariant (i) always holds. Hence, we only have to prove that Invariant (ii) is maintained during the loop and, hence, holds again at Line 16.

Let $f = S(d_1, \ldots, d_n)$ be the fact removed from new at Line 6. As $f \in \mathit{new}$ at the start of the round and $\mathsf{cost}(f) = x$, we can apply Invariant (ii)b to conclude $\mathsf{cost}(f) = \mathsf{optimal}(f)$. Hence, after adding f to done at Line 6, Invariant (ii)a applies to f.

Now consider any other fact $g \neq f$. We distinguish three cases:

1. $\mathsf{cost}(g) = \mathsf{optimal}(g)$ at the start of the loop. By either Invariant (ii)a or Invariant (ii)b, we have $g \in (\mathit{new} \cup \mathit{done})$. Due to Condition iii (Line 7), no facts $g \in \mathit{done}$ will be inspected during the $i + 1$-th loop. Likewise, due to the conditions on Line 9, no facts $g \in \mathit{new}$ with $\mathsf{cost}(g) = \mathsf{optimal}(g)$ will be updated during the $i + 1$-th loop. Hence, at the end of the loop, either Invariant (ii)a or Invariant (ii)b still applies to g.

2. $\mathsf{cost}(g) > \mathsf{optimal}(g)$ at the start of the loop and for all trees $\mathcal{T}$ representing a $\mathcal{C}$-deriv-simple explanation of fact g we either have $f \notin \mathcal{T}$ or there exist facts $g'' \in \mathcal{T}$ with $g'' \in \mathit{new}$, $\mathsf{cost}(g'') = \mathsf{optimal}(g'')$, and $g'' \neq f$. Consider any tree $\mathcal{T}$ representing a $\mathcal{C}$-deriv-simple explanation of fact g and any fact $g'' \in \mathcal{T}$ with $\mathsf{cost}(g'') = \mathsf{optimal}(g'')$ with $g'' \neq f$. By Invariant (ii), $g'' \in (\mathit{new} \cup \mathit{done})$ at the start of the loop. As $g'' \neq f$, $g'' \in (\mathit{new} \cup \mathit{done})$ at the end of the loop and all conditions put on $\mathcal{T}$ and g'' by Invariant (ii)c are maintained. Now choose such a $g'' \in \mathcal{T}$ with $\mathsf{cost}(g'') = \mathsf{optimal}(g'')$, $g'' \neq f$, and $g'' \in \mathit{new}$ at the start of the loop. At the end of the loop, we still have $g'' \in \mathit{new}$. By Invariant (i), $\mathsf{cost}(g'') \geq \mathsf{cost}(f) = x$. Hence, $\mathsf{cost}(g) > \mathsf{cost}(g'')$ and Invariant (ii)c still applies to g at the end of the $i + 1$-th loop.

3. $\mathsf{cost}(g) > \mathsf{optimal}(g)$ at the start of the loop and there exists a tree $\mathcal{T}$ representing a $\mathcal{C}$-deriv-simple explanation of fact g with, for all facts $g'' \in \mathcal{T}$, either $g'' \in \mathit{done}$ or $g'' = f$ or $g'' = g$. In this case, f is a child of the root of tree $\mathcal{T}$ (the fact g) and during execution of the $i+1$-th loop, the loop at Line 7 will consider the derivation represented by $\mathcal{T}$ (or any other $\mathcal{C}$-deriv-simple explanation involving f) and either adds g to *new* and *result* such that $\mathsf{cost}(g) = \mathsf{optimal}(g)$ (Lines 11–13) or updates the existing derivation for g in *new* and *result* such that $\mathsf{cost}(g) = \mathsf{optimal}(g)$ (Line 9). In both cases, Invariant (ii)b applies to g at the end of the $i+1$-th loop. □

Next, we consider the computation of $\mathcal{C}$-ddepth-simple explanations. It turns out that we can use the SemiNaive-Annotated algorithm of Figure 4 for this task: if we process newly-created facts via a normal *queue* (instead of a priority queue ordered on derivation complexity as SNA-PQ does), then we process newly-created facts in order of the depth of derivations used to create these facts (instead of the derivation complexity). We conclude the following:

Proposition 1. *Let R be the name of an n-ary rule in Datalog program $\langle B, D\rangle$. The pair $R(c_1, \ldots, c_n) \mapsto z$ is in the result of SemiNaive-Annotated($\langle B, D\rangle$) if and only if $R(c_1, \ldots, c_n)$ is in the evaluation of R. Furthermore, z is the root of a derivation tree that is a $\mathcal{C}$-ddepth-simple explanation of fact $R(c_1, \ldots, c_n)$.*

As Datalog queries are monotone, the base-fact complexity of any $\mathcal{C}$-base-simple explanation of a fact f in the evaluation of rule named R can be computed in a brute-force manner. Consider a Datalog program $\langle B, D\rangle$ such that f is in the evaluation of rule named R. Now let $B' \subseteq B$ be a minimum-sized subset of the base facts B such that fact f is in the evaluation of R with respect to the Datalog program $\langle B', D\rangle$. For any $\mathcal{C}$-base-simple explanation represented by tree $\mathcal{T}$ of fact f, we must have $\mathcal{C}\text{-base}(\mathcal{T}) = |B'|$. Furthermore, after finding base facts B', it is straightforward to find a $\mathcal{C}$-base-simple explanation for fact f, e.g., by using Algorithm SemiNaive-Annotated($\langle B', D\rangle$) to derive an explanation for f.

Likewise, we can compute any $\mathcal{C}$-uderiv-simple explanation of fact f by finding a minimum-sized subset of those derived facts D that appear in a derivation tree explaining f. To build an explanation out of D, we can use a variant of Algorithm SemiNaive-Annotated($\langle B', D\rangle$) that is restricted to only produce the derived facts in D.

6 The Complexity of Some Simple Explanations

In the previous section, we introduced brute-force methods to compute $\mathcal{C}$-base-simple and $\mathcal{C}$-uderiv-simple explanations. These methods are impractically slow for larger datasets. In the following, we will prove that it is unlikely that substantially faster algorithms exist. We do so by showing that a suitable *decision-version* of the problem of computing $\mathcal{C}$-base-simple and $\mathcal{C}$-uderiv-simple explanations is NP-hard by a reduction from the *vertex-cover problem*.

Definition 6. *Let $G = (V, E)$ be a graph with nodes V and undirected edges E (hence, $(m, n) \in E$ if and only if $(n, m) \in E$). A* vertex cover *of G is a subset $V' \subset V$ such that if $(m, n) \in E$ then either $m \in V'$ or $n \in V'$ (or both). Hence, V' contains at least one end point of every edge in the graph.*

The decision-version of the vertex-cover problem *is as follows: given a graph G and integer k, is there a vertex-cover of size k (or less).*

Lemma 1 ([27]). *The vertex-cover problem is NP-complete.*

Next, we formalize a suitable decision-version of finding a simple explanation.

Definition 7. *Let $\mathcal{C} \in \{\mathcal{C}\text{-base}, \mathcal{C}\text{-deriv}, \mathcal{C}\text{-uderiv}, \mathcal{C}\text{-ddepth}\}$. The* $\mathcal{C}$-simple explanation problem *is as follows: given a Datalog program, fact f, and integer k, is there an explanation $\mathcal{T}$ for the derivation of fact f with $\mathcal{C}(\mathcal{T}) \leq k$.*

To show that both the $\mathcal{C}$-base-simple explanation problem and the $\mathcal{C}$-uderiv-simple explanation problem are NP-hard, we will next construct a reduction of the vertex-cover problem to both the $\mathcal{C}$-base-simple explanation problem and the $\mathcal{C}$-uderiv-simple explanation problem.

Consider a graph $G = (V, E)$ with N undirected edges. Assume there is a minimal-sized vertex cover of G of size k. In our Datalog program, we represent each node $v \in V$ by the base fact $\mathit{Node}(v) \leftarrow \bullet$.

To simplify presentation, we choose a subset $U \subset E$ such that every edge in E has a single representation in U: for every undirected edge $(m, n) \in E$, either $(m, n) \in U$ or $(n, m) \in U$, but not both.

Let $\mathbb{I} = \{1, \ldots, N\}$ be the set of $|U|$ consecutive integers. We shall use $\mathbb{I}$ as *edge symbols* to identify edges and let $b : U \to \mathbb{I}$ be a bijection mapping each edge to a unique integer in $\mathbb{I}$. Let $(m, n) \in U$ be an edge and let $i = b((m, n)) \in \mathbb{I}$ be the integer representing (m, n). In our Datalog program, we represent this edge by two base facts $\mathit{EndPoint}(i, m) \leftarrow \bullet$ and $\mathit{EndPoint}(i, n) \leftarrow \bullet$.

Next, let $i, 1 \leq i < N$, be a non-last integer in $\mathbb{I}$ and let $j = i + 1$. Our Datalog program introduces the base fact $\mathit{SymbolOrder}(i, j) \leftarrow \bullet$ to establish this order on edge symbols.

Hence, there exists a unique *SymbolOrder*-labeled path from 1 to N that visits every edge symbol in $\mathbb{I}$. Now consider the Datalog derivation rules:

$$
\begin{aligned}
\mathit{IndirectNode}(v) &\leftarrow \mathit{Node}(v);\\
\mathit{CoveringNode}(v) &\leftarrow \mathit{EndPoint}(v, m) \wedge \mathit{IndirectNode}(m);\\
\mathit{OrderStep}(v, w) &\leftarrow \mathit{SymbolOrder}(v, w) \wedge \mathit{CoveringNode}(v);\\
\mathit{OrderVisit}(v, w) &\leftarrow \mathit{OrderStep}(v, w);\\
\mathit{OrderVisit}(v, w) &\leftarrow \mathit{OrderVisit}(v, u) \wedge \mathit{OrderStep}(u, w);\\
\mathit{OrderPath}(v, w) &\leftarrow \mathit{OrderVisit}(v, w) \wedge \mathit{CoveringNode}(w).
\end{aligned}
$$

We note that the fact *OrderPath*(1, N) represents a unique *SymbolOrder*-labeled path from 1 to N that visits every edge symbol in $i \in \mathbb{I}$. Furthermore, the explanation for fact *CoveringNode*(i) associates each edge symbol i on this path

with one of its end points (which is a node part of the vertex cover represented by the explanation).

Now consider a derivation tree $\mathcal{T}$ explaining the derivation of fact $\mathit{OrderPath}(1, N)$. This tree will consist of the following:

1. The derived fact $\mathit{OrderPath}(1, N)$.
2. The derived facts $\mathit{OrderVisit}(1, N)$ and $\mathit{CoveringNode}(N)$.
3. The derived facts $\mathit{OrderVisit}(1, i)$ and $\mathit{OrderStep}(i, i+1)$, for all $i, 3 \leq i < N$.
4. The derived facts $\mathit{OrderVisit}(1, 2)$ and $\mathit{OrderStep}(1, 2)$.
5. The base facts $\mathit{SymbolOrder}(i, i+1)$ and the derived facts $\mathit{CoveringNode}(i)$, for all $i, 1 \leq i < N$.
6. The base facts $\mathit{EndPoint}(i, n)$ for all $i, 1 \leq i \leq N$ with $n \in V$.
7. For each of these $\mathit{EndPoint}(i, n)$, a derived fact $\mathit{IndirectNode}(n)$ and base fact $\mathit{Node}(n)$.

Let V' be the set of nodes used in the facts $\mathit{EndPoint}(i, n)$, $i, 1 \leq i \leq N$, used in $\mathcal{T}$. By construction, V' is a vertex cover. Note that $\mathcal{T}$ includes derived facts $\mathit{IndirectNode}(n)$ and base fact $\mathit{Node}(n)$ if and only if $n \in V'$. Hence, the size $k = |V'|$ of vertex cover V' is equivalent to the number of derived facts with head *IndirectNode* in $\mathcal{T}$ and is also equivalent to the number of base facts with head *Node* in $\mathcal{T}$.

Due to the above breakdown of explanation $\mathcal{T}$, we can conclude that $\mathcal{T}$ includes $(3N - 1) + k$ derived facts (one with head *OrderPath*, $N - 1$ with head *OrderVisit*, $N - 1$ with head *OrderStep*, N with head *CoveringNode*, and k with head *IndirectNode*), and that $\mathcal{T}$ includes $2(N - 1) + k$ base facts ($N - 1$ with head *SymbolOrder*, $N - 1$ with head *EndPoint*, and k with head *Node*).

Theorem 3. *The $\mathcal{C}$-base-simple explanation problem and the $\mathcal{C}$-uderiv-simple explanation problem are NP-complete, even for Datalog programs in which all heads have at-most two symbols and all bodies of derivation rules have at-most two terms.*[1]

Proof. First, we prove that both problems are NP-hard. Consider the vertex-cover problem for undirected graph $G = (V, E)$ and integer k. Using the above Datalog construction, there exists a vertex-cover of size k if and only if there exists an explanation $\mathcal{T}$ for fact $\mathit{EdgePath}(1, N)$ with $\mathcal{C}\text{-base}(\mathcal{T}) = 2(N - 1) + k$ or with $\mathcal{C}\text{-uderiv}(\mathcal{T}) = (3N - 1) + k$.

Next, we prove both problems are in NP. To do so, we adapt the brute-force methods for computing $\mathcal{C}$-base-simple explanations and $\mathcal{C}$-uderiv-simple explanations outlined at the end of Section 5 to show how, for any integer k, one can represent solutions to the $\mathcal{C}$-base-simple explanation problem and the $\mathcal{C}$-uderiv-simple explanation problem using sets of at-most k base facts

[1] Hence, the *data complexity* (the complexity in terms of only the number of base facts) [33] for both the $\mathcal{C}$-base-simple explanation problem and the $\mathcal{C}$-uderiv-simple explanation problem is already NP-hard, even though the data complexity of normal Datalog query evaluation, for Algorithm SemiNaive-Annotated, and for Algorithm SNA-PQ is polynomial.

($\mathcal{C}$-base-simple explanation problem) or at-most k derived facts ($\mathcal{C}$-uderiv-simple explanation problem) and how one can certify such a representation efficiently using the SemiNaive-Annotated algorithm.

7 Discussion

In this paper, we have established *four* types of simple explanations, two of which ($\mathcal{C}$-deriv-simple and $\mathcal{C}$-ddepth-simple explanations) can be computed efficiently in practice. The ability to efficiently compute a simple explanation does *not* guarantee that the resulting explanations are practically usable, however: even simple explanations can become very large.

The *size* of simple explanations for Datalog programs depends on the complexity of the derivation rules and the patterns present in the base facts. For many common types of Datalog queries, reasonable upper bounds on the size of simple explanations exist, however:

1. all non-recursive programs will have simple explanations whose size is upper bounded by the number of derivation rules; and
2. all simple recursive programs (e.g., transitive closures, reachability, and other constructions typically used in navigational graph queries [3, 10, 12, 23, 24]) will have simple explanations whose size is upper bounded by the number of base facts.

Due to results on simple explanations for context-free path queries [23, 24], no guarantees for small simple explanations can be provided for Datalog programs with more complex forms of recursion, however.

8 Related Work

There is an abundant literature on data provenance [9], provenance for Datalog [7, 8, 15, 17, 30], explaining graph queries [24], explaining SHACL [14], and the concept of short explanations [24, 26]. There is only limited work toward explaining Datalog queries, however. We provide a brief overview.

Explaining graph queries. Several recent works have focused on explaining navigation-based graph queries using either paths in the graph or subgraphs of the graph, e.g., explanations for regular path queries or context-free path queries [4, 13, 19, 22, 24, 25, 31, 32]. These types of graph queries are a fragment of the type of queries that can be expressed via Datalog, e.g., the explanations of context-free path queries [24] are a very restricted example of explanations with low derivation complexity. We vastly generalize the limited results of these works by considering explanations in Datalog, a much more expressive query language. Furthermore, we look at multiple fundamental types of *short and simple* explanations instead of focusing on a single path-based or subgraph-based semantics for explanations.

Why-provenance for Datalog. Typical works on provenance for Datalog focus on providing thorough explanation of facts f by explaining all the different ways

in which one can derive f [6–8, 15, 17, 30]. For example, the explanation of a fact f is its support [7]: all different subsets of the set of base facts that can be used to derive f. As such a thorough explanation is too large for practical use, the focus of several works [15, 17, 30] is on finding a practically-computable and representative small-enough sample of this thorough explanation. We take a different approach by only looking for a *single* explanation that is short and simple (according to fundamental notions of complexity of explanations).

Of these recent works on Datalog provenance, the work by Calautti et al. [7] is closest to our approach: in their work, the provenance of a fact f consist of the set $S(f)$ of all supports for f. The smallest support in $S(f)$ corresponds to the notion of a $\mathcal{C}$-base-simple explanation that we study. They show that obtaining the provenance of fact f involves solving NP-complete decision problems. Our results on the complexity of $\mathcal{C}$-base-simple explanations (Theorem 3) are more general, however: $\mathcal{C}$-base-simple explanations for fact f are simpler than the provenance for f and our results already hold for simple derivations involving only Datalog rules with at-most two terms and an arity at-most-2. Furthermore, our work considers not only $\mathcal{C}$-base-simple explanations, but also other fundamental notions of short and simple explanations.

9 Conclusions and Future Work

Toward improving *explanations* for the results of complex recursive queries, we formally studied short and simple explanations of Datalog derivations. We introduced four distinct notions to measure the complexity of explanations (base fact, derivation, unique derivation, and derivation depth complexity) and introduce four corresponding types of *simple* explanations that minimize one of these complexity measures. In addition, we provided efficient evaluation algorithms to compute explanations with minimal derivation or derivation depth complexity. Finally, we showed that computing explanations with minimal base fact complexity or unique derivation complexity involves solving NP-complete decision problems and, hence, is inherently hard.

We are encouraged by our results: they indicate that *simple explanations* are practically possible with respect to derivation and derivation depth complexity. Hence, the next step is to engineer practically-efficient algorithms to produce such explanations. The algorithms in our paper are only suitable for select use-cases, however: they are *bottom-up algorithms* that are notoriously costly when one is only interested in a subset of the data (e.g., only facts for a few rule names or only facts matching a given selection). To provide practical evaluation algorithms for such applications, we are interested in the development of *top-down* and *goal-oriented* algorithms, e.g., based on magic set evaluation. Based on the results in this paper, we believe adaptations of existing efficient Datalog evaluation techniques are achievable.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases: The Logical Level. Addison-Wesley, 1st edn. (1995)
2. Alrabbaa, C., Baader, F., Borgwardt, S., Koopmann, P., Kovtunova, A.: Finding good proofs for description logic entailments using recursive quality measures. In: Automated Deduction – CADE 28. pp. 291–308. Springer (2021). https://doi.org/10.1007/978-3-030-79876-5_17
3. Barceló, P.: Querying graph databases. In: Proceedings of the 32nd Symposium on Principles of Database Systems. pp. 175–188. ACM (2013)
4. Barrett, C., Jacob, R., Marathe, M.: Formal-language-constrained path problems. SIAM Journal on Computing **30**(3), 809–837 (2000)
5. Bellomarini, L., Benedetto, D., Gottlob, G., Sallinger, E.: Vadalog: A modern architecture for automated reasoning with large knowledge graphs. Inf. Syst. **105**(C) (2022). https://doi.org/10.1016/j.is.2020.101528
6. Bourgaux, C., Bourhis, P., Peterfreund, L., Thomazo, M.: Revisiting semiring provenance for datalog. In: Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning. pp. 91–101 (2022). https://doi.org/10.24963/kr.2022/10
7. Calautti, M., Livshits, E., Pieris, A., Schneider, M.: The complexity of why-provenance for datalog queries (2023), https://arxiv.org/abs/2303.12773
8. Calautti, M., Livshits, E., Pieris, A., Schneider, M.: Computing the why-provenance for datalog queries via SAT solvers. In: Proceedings of the AAAI Conference on Artificial Intelligence. pp. 10459–10466 (2024). https://doi.org/10.1609/aaai.v38i9.28914
9. Cheney, J., Chiticariu, L., Tan, W.C.: Provenance in databases: Why, how, and where. Foundations and Trends in Databases **1**(4), 379–474 (2009)
10. Consens, M.P., Mendelzon, A.O.: GraphLog: A visual formalism for real life recursion. In: Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. pp. 404–416. ACM (1990)
11. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, Fourth Edition. The MIT Press, 4th edn. (2022)
12. Cruz, I.F., Mendelzon, A.O., Wood, P.T.: A graphical query language supporting recursion. In: Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data. pp. 323–330. ACM (1987)
13. Cucumides, T., Reutter, J., Vrgoč, D.: Size bounds and algorithms for conjunctive regular path queries. In: 26th International Conference on Database Theory (ICDT 2023). Leibniz International Proceedings in Informatics (LIPIcs), vol. 255, pp. 13:1–13:17. Schloss Dagstuhl (2023). https://doi.org/10.4230/LIPIcs.ICDT.2023.13
14. Delva, T., Dimou, A., Jakubowski, M., den Bussche, J.V.: Data provenance for SHACL. In: Proceedings 26th International Conference on Extending Database Technology. pp. 285–297. OpenProceedings.org (2023). https://doi.org/10.48786/EDBT.2023.23
15. Deutch, D., Gilad, A., Moskovitch, Y.: Selective provenance for datalog programs using top-k queries. Proceedings of the VLDB Endowment **8**(12), 1394–1405 (2015). https://doi.org/10.14778/2824032.2824039
16. Dowling, W.F., Gallier, J.H.: Linear-time algorithms for testing the satisfiability of propositional horn formulae. The Journal of Logic Programming **1**(3), 267–284 (1984). https://doi.org/10.1016/0743-1066(84)90014-1
17. Elhalawati, A., Krötzsch, M., Mennicke, S.: An existential rule framework for computing why-provenance on-demand for datalog. In: Rules and Reasoning: 6th

International Joint Conference on Rules and Reasoning, RuleML+RR 2022. pp. 146–163. Springer-Verlag (2022). https://doi.org/10.1007/978-3-031-21541-4_10
18. Fuhr, N.: Probabilistic datalog–a logic for powerful retrieval methods. In: Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 282–290. ACM (1995). https://doi.org/10.1145/215206.215372
19. García, R., Angles, R.: An algebra for path manipulation in graph databases. In: Advances in Databases and Information Systems. pp. 61–74. Springer International Publishing (2022)
20. Greco, S., Molinaro, C.: Datalog and Logic Databases. Springer (2016). https://doi.org/10.1007/978-3-031-01854-1
21. Green, T.J., Huang, S.S., Loo, B.T., Zhou, W.: Datalog and recursive query processing. Foundations and Trends in Databases **5**(2), 105–195 (2013). https://doi.org/10.1561/1900000017
22. Grigorev, S., Ragozina, A.: Context-free path querying with structural representation of result. In: Proceedings of the 13th Central & Eastern European Software Engineering Conference in Russia. ACM (2017). https://doi.org/10.1145/3166094.3166104
23. Hellings, J.: Conjunctive context-free path queries. In: Proceedings of the 17th International Conference on Database Theory (ICDT). pp. 119–130. OpenProceedings.org (2014). https://doi.org/10.5441/002/icdt.2014.15
24. Hellings, J.: Explaining results of path queries on graphs: Single-path results for context-free path queries. Information Systems **128**, 102475 (2025). https://doi.org/10.1016/j.is.2024.102475
25. Hofman, P., Martens, W.: Separability by short subsequences and subwords. In: 18th International Conference on Database Theory. Leibniz International Proceedings in Informatics (LIPIcs), vol. 31, pp. 230–246. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (2015)
26. Jaakkola, R., Janhunen, T., Kuusisto, A., Rankooh, M.F., Vilander, M.: Short boolean formulas as explanations in practice. In: Logics in Artificial Intelligence. pp. 90–105. Springer (2023). https://doi.org/10.1007/978-3-031-43619-2_7
27. Karp, R.M.: Reducibility among Combinatorial Problems, pp. 85–103. Springer (1972). https://doi.org/10.1007/978-1-4684-2001-2_9
28. Ketsman, B., Koutris, P.: Modern datalog engines. Foundations and Trends in Databases **12**(1), 1–68 (2022). https://doi.org/10.1561/1900000073
29. Lanzinger, M., Sferrazza, S., Gottlob, G.: MV-Datalog+/-: Effective rule-based reasoning with uncertain observations (extended abstract). In: Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence. pp. 6447–6451. International Joint Conferences on Artificial Intelligence Organization (8 2023). https://doi.org/10.24963/ijcai.2023/718
30. Lee, S., Ludäscher, B., Glavic, B.: Approximate summaries for why and why-not provenance. Proceedings of the VLDB Endowment **13**(6), 912–924 (2020). https://doi.org/10.14778/3380750.3380760
31. Martens, W., Niewerth, M., Popp, T., Rojas, C., Vansummeren, S., Vrgoč, D.: Representing paths in graph database pattern matching. Proc. VLDB Endow. **16**(7) (2023). https://doi.org/10.14778/3587136.3587151
32. Terekhov, A., Khoroshev, A., Azimov, R., Grigorev, S.: Context-free path querying with single-path semantics by matrix multiplication. In: Proceedings of the 3rd Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA). ACM (2020). https://doi.org/10.1145/3398682.3399163

33. Vardi, M.Y.: The complexity of relational query languages (extended abstract). In: Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing. pp. 137–146. ACM (1982). https://doi.org/10.1145/800070.802186