Efficient Provenance-Aware Querying of Graph Databases with Datalog

Yann Ramusat, Silviu Maniu, Pierre Senellart

Efficient Provenance-Aware Querying of Graph Databases with Datalog

Yann Ramusat
DI ENS, ENS, PSL University, CNRS, Inria
Paris, France
yann.ramusat@ens.fr

Silviu Maniu
Université Paris-Saclay, LISN, CNRS
Gif-sur-Yvette, France
silviu.maniu@lisn.upsaclay.fr

Pierre Senellart
DI ENS, ENS, PSL University, CNRS, Inria & IUF
Paris, France
pierre@senellart.com

ABSTRACT

We establish a translation between a formalism for dynamic programming over hypergraphs and the computation of semiring-based provenance for Datalog programs. The benefit of this translation is a new method for computing the provenance of Datalog programs for specific classes of semirings, which we apply to provenance-aware querying of graph databases. Theoretical results and practical optimizations lead to an efficient implementation using Soufflé, a state-of-the-art Datalog interpreter. Experimental results on real-world data suggest this approach to be efficient in practical contexts, competing with dedicated solutions for graphs.

CCS CONCEPTS

• **Information systems** → **Data provenance**; • **Theory of computation** → **Data provenance**.

KEYWORDS

Datalog, Provenance, Semirings, Graph Databases, Dynamic Programming, Transportation Networks

1 INTRODUCTION

Data provenance [3] is meta-information that is kept along and tracked about data throughout its life cycle, and which is propagated to query results during query evaluation; applications of such information include traceability, explainability, or uncertainty management [21]. A notion of provenance for Datalog queries was introduced by Green, Karvounarakis, and Tannen [9]. Based on the algebraic structure of *semirings* to encode additional meta-information about query results, it extends the notion of semiring provenance for the positive fragment of the relational algebra, also introduced in [9]. The *full provenance* of a Datalog program (i.e., the provenance associated to each *derived tuple*) is expressed as a *system of equations* over an ω*-continuous* semiring. This forms a "computational" notion of provenance, where operations (and queries) over provenance values are permitted.

A recent line of work has adapted this provenance model for simple navigational queries (*regular path queries* [2] or *RPQ*s) over graph databases [18, 19]. Several provenance-aware algorithms have been proposed, and a taxonomy of semiring classes, based on their properties, has been established in [19]. Those works aim at identifying, for a set of important semiring classes, the most appropriate algorithm for provenance-aware querying, enabling real-world applications.

The aim of the current work is to extend these approaches to queries that go beyond the simple class of RPQs, relying on the rich literature around Datalog provenance to provide better solutions for provenance computations over graph databases. Our objective is to obtain new effective solutions to practical scenarios (i.e., real transportation networks over large areas). In the process, we generalize the methods for provenance computation over graph databases of [19] to Datalog. Our main motivation is thus is to allow new opportunities for querying graph databases in the presence of provenance information, as Datalog is significantly more expressive than RPQs. Given the fact that RPQs are expressible in Datalog, we can therefore either compose or take the union of several RPQs. Whereas RPQs only select pair of vertices, with Datalog we can go beyond binary output relations; this allows to retrieve, for instance, paths that are restricted to pass through another node.

Our contributions can be organized into three parts. We first establish a correspondence between dynamic programming over hypergraphs (as introduced in [10] under the name of *weighted hypergraphs*) and the *proof-theoretic* definition of the provenance for Datalog programs. We provide both-way translations and characterize for which class of semirings the *best-weight derivation* in the hypergraph corresponds to the provenance of the initial Datalog program.

The translation we thus introduced permits us to obtain a version of Knuth's generalization of Dijkstra's algorithm to the *grammar problem* [13], adapted to the case of Datalog provenance computation. In the special setting where all hyperedges are of arity 1, we obtain the classical notion of semiring-based provenance for graph databases [18]. In the general setting, the algorithm steadily generalizes to Datalog the adapted Dijkstra's algorithm from [18], under the same assumptions on the properties of the underlying semiring. Such algorithm is unlikely to be efficient as-is in practical contexts. The main issue is closely related to the inefficiency of basic Datalog evaluation: many computations of facts (provenance values) have already been deduced, leading to redundant computations. Nevertheless, we show that the *semi-naïve* evaluation strategy for Datalog is also applicable in our setting. An added advantage is that it facilitates extending existing Datalog solvers to provenance annotations and their computation.

We implement our strategy by extending Soufflé [11], a state-of-the-art Datalog interpreter. We apply our solution to the computation of the provenance of various graph queries (translated into Datalog programs) on several real-world graph datasets. Experiments witness that the performance of the implementation competes with previous dedicated solutions specific to graph databases, while allowing much more expressive queries than the works of [18, 19]: RPQs only select pairs of vertices joinable by a path whose label belongs to a given regular language. We focus our experiments

on *graph patterns* that are combinations and/or unions of RPQs, using the expressive power of Datalog. We show that for this wider class of navigational queries, our method performs well in practice. Notably, the ratio between the running time of our approach compared to plain Datalog evaluation (not tracking the provenance) is experimentally bounded by a small constant factor. Moreover, in case of large output DB size, the average number of output tuples processed by seconds is up to a million, permitting a reasonable running time in practice for large datasets.

The paper is organized as follows. We start by introducing in Section 2 basic concepts on semirings and we recall the definition of provenance for Datalog programs. We formulate and prove in Section 3 the correspondence between weighted hypergraphs and semiring-based provenance for Datalog programs. In Section 4, we present the adapted version of Knuth's algorithm for the grammar problem and discuss theoretical aspects of its optimization. We then dive into the practical aspects of its implementation using Soufflé, and present in Section 5 experimental results witnessing the efficiency of our approach for practical scenarios. We finally discuss in Section 6 the related work. For space reasons, proofs are deported to an appendix.

2 BACKGROUND

In the following, we recall basic concepts of semiring theory underlying the optimization techniques we provide in this paper. For more background on the theory and applications of semirings, examples of relevant semirings, as well as references to the literature on advanced notions of semiring theory, see [19]. We mostly follow the definitions from [19] and also highlight notions that occur under different names depending on the application domain.

Definition 1 (Semiring). A *semiring* is an algebraic structure $(S, \oplus, \otimes, \bar{0}, \bar{1})$ where S is some set, $\oplus$ and $\otimes$ are binary operators over S, and $\bar{0}$ and $\bar{1}$ are elements of S, satisfying the following axioms:

- $(S, \oplus, \bar{0})$ is a *commutative monoid*: $(a \oplus b) \oplus c = a \oplus (b \oplus c)$, $a \oplus b = b \oplus a$, $a \oplus \bar{0} = \bar{0} \oplus a = a$;
- $(S, \otimes, \bar{1})$ is a *monoid*: $(a \otimes b) \otimes c = a \otimes (b \otimes c)$, $\bar{1} \otimes a = a \otimes \bar{1} = a$;
- $\otimes$ distributes over $\oplus$: $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$;
- $\bar{0}$ is an annihilator for $\otimes$: $\bar{0} \otimes a = a \otimes \bar{0} = \bar{0}$.

A semiring is *commutative* if for all $a, b \in S$, $a \otimes b = b \otimes a$. A semiring is *idempotent* if for all $a \in S$, $a \oplus a = a$. For an idempotent semiring we can introduce the *natural order* defined by $a \preccurlyeq b$ iff $a \oplus b = a$.[1] Note that this order is compatible with the two binary operations of the semiring: for all $a, b, c \in S$, $a \preccurlyeq b$ implies $a \oplus c \preccurlyeq b \oplus c$ and $a \otimes c \preccurlyeq b \otimes c$. This is also called the *monotonicity* property.

An important property is that of *k-closedness* [17], i.e., a semiring is k-closed if: $\forall a \in S$, $\bigoplus_{i=0}^{k+1} a^i = \bigoplus_{i=0}^{k} a^i$. Here, by a^i we denote the repeated application of the $\otimes$ operation i times, i.e., $a^i = \underbrace{a \otimes a \otimes \cdots \otimes a}_{i}$. 0-closed semirings (i.e., those in which $\forall a \in S, \bar{1} \oplus a = \bar{1}$) have also been called *absorptive*, *bounded*, or *simple* depending on the literature. Note that any 0-closed semiring is idempotent (indeed, $a \oplus a = a \otimes (\bar{1} \oplus \bar{1}) = a \otimes \bar{1} = a$) and therefore admits a natural order.

Huang [10] introduces the notion of *superiority* of a semiring S with respect to a partial order $\leqslant$, defined by: $\forall a, b \in S\, a \leqslant a \otimes b,\ b \leqslant a \otimes b$. The natural order satisfies this notion for 0-closed semirings:

Lemma 2. *Let S be an idempotent semiring and $\leqslant$ the natural order over S. Then S is* superior *with respect to $\leqslant$ if and only if S is 0-closed.*

An easier way of understanding natural order in 0-closed semirings is to note that for any idempotent semiring S, $\bar{0}$ is the greatest element ($\forall a \in S, a \oplus 0 = a \preccurlyeq \bar{0}$) while, if the semiring is also 0-closed (i.e., *bounded*), $\bar{1}$ is the smallest ($\forall a \in S$, $\bar{1} \oplus a = \bar{1}$ so $\bar{1} \preccurlyeq a$). Thus a bounded semiring S verifies $\bar{1} \preccurlyeq a \preccurlyeq \bar{0}$ for all $a \in S$.

Definition 3 (ω-Continuous semiring). An idempotent semiring $(S, \oplus, \otimes, \bar{0}, \bar{1})$ is *ω-continuous* if:

(1) $(S, \geqslant)$ is an *ω-complete* partial order, i.e., the infimum $\inf_{i \in \mathbb{N}} a_i$ of any infinite chain $a_0 \geqslant a_1 \geqslant \dots$ exists in S.
(2) both addition and multiplication are *ω-continuous* in both arguments, i.e., for all $a \in S$ and infinite chain $a_0 \geqslant a_1 \geqslant \dots$, $(a \oplus \inf_{i \in \mathbb{N}} a_i) = \inf_{i \in \mathbb{N}} (a \oplus a_i)$, $(a \otimes \inf_{i \in \mathbb{N}} a_i) = \inf_{i \in \mathbb{N}} (a \otimes a_i)$, $(\inf_{i \in \mathbb{N}} a_i) \otimes a = \inf_{i \in \mathbb{N}} (a_i \otimes a)$.

In such semirings we can define countable sums:

$$\bigoplus_{n \in \mathbb{N}} a_n = \inf_{m \in \mathbb{N}} \bigoplus_{i=0}^{m} a_i.$$

A *system of fixpoint equations* over an ω-continuous semiring S is a finite set of equations: $X_1 = f_1(X_1, X_2, \dots, X_n)$, $\dots$, $X_n = f_n(X_1, X_2, \dots, X_n)$, where $X_1, \dots, X_n$ are variables and $f_1, \dots f_n$ are polynomials with coefficients in S. We extend the notion of natural order from semiring elements to tuples of semiring elements by simply considering the product order. We then have the following on solutions of a system of equations over an ω-continuous semiring:

Theorem 4 (Theorem 3.1 of [15]). *Every system of fixpoint equations $\mathbf{X} = f(\mathbf{X})$ over a commutative ω-continuous semiring has a greatest solution $\mathrm{gfp}(f)$ w.r.t. $\leqslant$, and $\mathrm{gfp}(f)$ is equal to the infimum of the Kleene sequence: $\mathrm{gfp}(f) = \inf_{m \in \mathbb{N}} f^m(\bar{0})$.*

We now recall some basics about the Datalog query language and refer to [1] for more details. A *Datalog rule* is of the form $R(\vec{x}) \coloneq R_1(\vec{x_1}), \dots, R_n(\vec{x_n})$ with R's representing relations of a given arity and the $\vec{x}$'s tuples of variables of corresponding arities. Variables occurring on the left-hand side, the *head* of the rule, are required to occur in at least one of the atoms on the right-hand side, the *body* of the rule. A *Datalog program* is a finite set of Datalog rules. We call *fact* a rule with an empty body and variables replaced by constants. We divide relations into *extensional* ones (which can only occur as head of a fact, or in rule bodies) and *intensional* ones (which may occur as heads of a non-fact rule). The set of extensional facts is called the *extensional database* (EDB). We distinguish one particular relation occurring in the head of a rule, the *output predicate* of the Datalog program. We refer to [1] for the semantics of such a program and the notion of *derivation tree*.

[1] There are unfortunately two definitions of natural order commonly found in the literature; we use here that found in [10, 17] which matches the standard order on the tropical semiring; other works [9, 15, 19] define it as the reverse order. Our choice obviously has some impacts: in particular, when defining Datalog provenance, we need greatest fixpoints in lieu of the least fixpoints used in [9, 15].

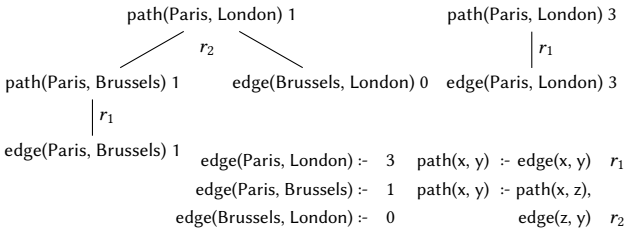

Figure 1: Derivation trees along their weights for the fact path(Paris, London) using the transitive closure Datalog program over the tropical semiring with an EDB **containing** 3 **facts.**

There are two ways of defining the *provenance* of a Datalog program q with output predicate G over an ω-continuous semiring. We can first base this definition on the proof-theoretic definition of standard Datalog:

Definition 5 (Proof-theoretic definition for Datalog provenance [9]). Let $(S, \oplus, \otimes, \bar{0}, \bar{1})$ be a commutative ω-continuous semiring and q a Datalog program with output predicate S and such that all extensional facts $R(t')$ are annotated with an element of S, denoted as $\text{prov}_R^q(t')$. Then the *provenance* of $G(t)$ for q, where $G(t)$ is in the output of q, is defined as:

$$\text{prov}_G^q(t) = \bigoplus_{\tau \text{ yields } t} \left(\bigotimes_{t' \in \text{leaves}(\tau)} \text{prov}_R^q(t') \right).$$

The first sum ranges over all the derivation trees of the fact t (see Figure 1 for examples of derivation trees), the second sum ranges over all leaves of the tree (extensional facts). This definition describes how the provenance propagates across the deduction process given an initial assignment of provenance weights to the extensional relations of q, prov_R^q.

Example 6. The *tropical semiring* is $(\mathbb{R}^+ \cup \{\infty\}, \min, +, \infty, 0)$. It is a 0-closed (and thus idempotent) ω-continuous semiring. We show in Figure 1 an example Datalog program (bottom right) with tropical semiring annotations on extensional edge facts, as well as the (only) two derivation trees of the fact path(Paris, London) along their weight. This witnesses that the provenance of path(Paris, London) is $\min(1 + 0, 3) = 1$.

Since some tuples can have infinitely many derivations, the Datalog semantics of Definition 5 cannot be used as an algorithm. As pointed out in [9], it is possible instead to use a fixpoint-theoretic definition of the provenance of a Datalog query q: introduce a fresh variable for every possible *intensional* tuple (i.e., every possible *ground atom*), and produce for this variable an equation that reflects the *immediate consequence operator* T_q – *extensional* facts appearing as their semiring annotations in these equations. This yields a system of fixpoint equation f_q. The provenance of $G(t)$ for q is now simply the value of the variable corresponding to $G(t)$ in $\text{gfp}(f_q)$.

The fixpoint-theoretic definition directly yields an algorithm, albeit a very inefficient one because of the need of generating a rule for every intensional tuple.

3 DATALOG PROVENANCE AND DYNAMIC PROGRAMMING OVER HYPERGRAPHS

We now show how to convert a Datalog program into a weighted hypergraph (as introduced in [10]) and characterize the semirings where the best-weight derivation in the hypergraph corresponds to the provenance for the initial Datalog program, mimicking the proof-theoretic definition. We first recall basic definitions and notation related to hypergraphs.

Definition 7 (Weighted hypergraph [10]). Given a semiring S, a *weighted hypergraph* on S is a pair $H = \langle V, E \rangle$, where V is a finite set of vertices and E is a finite set of hyperedges, and each element $e \in E$ is a triple $e = \langle h(e), \mathrm{T}(e), f_e \rangle$ with $h(e) \in V$ its *head* vertex, $\mathrm{T}(e) \in V$ an ordered list of *tail* vertices and f_e a *weight function* from $S^{|\mathrm{T}(e)|}$ to S.

We note $|e| = |\mathrm{T}(e)|$ the *arity* of a hyperedge. If $|e| = 0$, we say e is nullary and then $f_e()$ is a constant, an element of the semiring; we assume there exists at most one nullary edge for a given vertex. In that case, $v = h(e)$ is called a *source vertex* and we note $f_e()$ as f_v. The *arity of a hypergraph* is the maximum arity of any hyperedge.

The *backward-star* $\mathrm{BS}(v)$ of a vertex v is the set of incoming hyperedges $\{e \in E \mid h(e) = v\}$. The *graph projection* of a hypergraph $H = \langle V, E \rangle$ is a directed graph $G = (V, E')$ where $E' = \{(u, v) \mid \exists e \in \mathrm{BS}(v), u \in \mathrm{T}(e)\}$. A hypergraph H is acyclic if its graph projection G is acyclic; then a topological ordering of H is an ordering of V that is a topological ordering of G.

With these definitions in place, we can encode a Datalog program with semiring annotations as a weighted hypergraph in a straightforward manner:

Definition 8 (Hypergraph representation of a Datalog program). Given a Datalog program q described with a set of rules $\{q_1, \ldots, q_n\}$ and the semiring S used for annotations, we define the *weighted hypergraph representation of* q as $H_q = \langle V_q, E_q \rangle$ with V_q being all ground atoms and, for each instantiation of a rule $t(\vec{x}) \leftarrow r_1(\vec{x_1}), \ldots, r_n(\vec{x_n})$, a corresponding edge $\langle t(\vec{x}), (r_1(\vec{x_1}), \ldots, r_n(\vec{x_n})), \otimes \rangle$. For a fact $R(\vec{x}) \in \text{EDB}(q)$ we add a nullary edge e with $h(e) = R(\vec{x})$ and $f_e = \text{prov}_R^q(\vec{x})$.

The notion of *derivations* is the hypergraph counterpart to paths in graph. We recall the definition of derivations and we define it in a way that is reminiscent of Datalog-related notions.

Definition 9 (Derivation in hypergraph [10]). We recursively define a *derivation* D of a vertex v in a hypergraph H (as a pair formed of a hyperedge and a list of derivations), its size $|D|$ (a natural integer) and its weight $w(D)$ (a semiring element) as follows:

- If $e \in \mathrm{BS}(v)$ with $|e| = 0$, then $D = \langle e, \langle\rangle \rangle$ is a derivation of v, $|D| = 1$, and $w(D) = f_e()$.
- If $e \in \mathrm{BS}(v)$ with $|e| \geqslant 0$, D_i is a derivation of $T_i(e)$ for $i = 1 \ldots |e|$, then $D = \langle e, \langle D_1 \cdots D_{|e|} \rangle \rangle$ is a derivation of v, $|D| = 1 + \sum_{i=1}^{|e|} |D_i|$, $w(D) = f_e(w(D_1), \ldots, w(D_{|e|}))$.

We note $\mathcal{D}_H(v)$ the set of derivations of v in H.

When modeling Datalog provenance in a semiring S as weighted hypergraphs on S, all non-source weight functions are bound to the $\otimes$ operation of the semiring. Note that, if S is idempotent, the

natural order on S induces an ordering on derivations: $D \leqslant D'$ if $w(D) \leqslant w(D')$.

We now show that in this formalism, the Datalog provenance of an output predicate can be understood as the best-weight for the corresponding vertex in the hypergraph.

Definition 10 (Best-weight [10]). The *best-weight* $\delta_H(v)$ of a vertex v of a hypergraph H on a semiring $(S, \oplus, \otimes, \bar{0}, \bar{1})$ is the weight of the best derivation of v:

$$\delta_H(v) = \begin{cases} f_v & \text{if v is a source vertex;} \\ \bigoplus_{D \in \mathcal{D}_H(v)} w(D) & \text{otherwise.} \end{cases}$$

The best-weight generally requires additional properties of either the hypergraph or the semiring to be well-defined. Acyclicity for the hypergraph is a sufficient condition. Existence of an infinitary summation operator in the semiring extending $\oplus$, guaranteed in ω-continuous semirings, is also a sufficient condition. To guarantee semantics compatible with the intuitive meaning of provenance, a more restrictive sufficient condition is for the semiring to be a *c-complete star-semiring* [14], see [19] for details.

We can now show that Datalog provenance can be computed through the formalism of weighted hypergraphs. Let us start with a lemma exhibiting a one-to-one mapping between derivations in the hypergraph and proofs in Datalog.

Lemma 11. *For any Datalog query q and grounding of an atom v of q, there is a bijection between $\mathcal{D}_{Hq}(v)$ and $\{\tau \mid \tau \text{ yields } v\}$.*

We then show that the weight of each derivation of a tuple is equal to the corresponding proof tree weight in Datalog.

Lemma 12. *For any Datalog query q and grounding of an atom v of q, for any derivation D of v in H_q we have*

$$w(D) = \bigotimes_{t' \in leaves(\tau_D)} \text{prov}_R^q(t')$$

where τ_D is the proof tree corresponding to D in the bijection given by Lemma 11.

Finally, we obtain:

Theorem 13. *Let t be a tuple of a Datalog program q with output predicate G and H_q its hypergraph representation, then* $\text{prov}_G^q(t) = \delta_{H_q}(G(t))$.

4 BEST-FIRST METHOD

Knuth [13] generalized the Dijkstra algorithm to what he calls the *grammar problem* (i.e., finding the *best-weight derivation* from a given non-terminal, where each terminal has a specific weight and each rule comes with an associated weight function). This has been identified as corresponding to the search problem in a monotonic superior hypergraph – i.e., for each $e \in H$, f_e is monotone and superior in each argument (see Table 3 in [10]). We showed in Lemma 2 that superiority corresponds to 0-closedness in semirings with natural orders. The definition of the grammar problem assumes a total order on weights as the weights are real numbers. In the special case where all hyperedges are of arity 1 (and all weight functions bound to $\otimes$), we obtain the classical notion of semiring-based provenance for graph databases [18]. Thus, Knuth's algorithm can be seen as a generalization to hypergraphs (and therefore, by the results of the previous section, to Datalog provenance computation) of the modified Dijkstra algorithm from [18], working on 0-*closed totally-ordered semirings*, which are generalizations of the tropical semiring.

Optimized version of Best-First method. In the original paper of Knuth [13], the question of efficient construction of the set of candidate facts for the extraction of the minimal-valued fact is not dealt with. A lot of redundant work may be carried out if the implementation is not carefully designed.

In the following, we show how to obtain a ready-to-be-implemented version incorporating ideas from the *semi-naïve* evaluation of Datalog programs. Semi-naïve evaluation of Datalog, as described in [1, Chapter 13] introduces a number of ideas aiming at improving the efficiency of the *naïve* Datalog evaluation method; we show how to leverage them in our setting.

The *naïve* evaluation of a Datalog program q processes iteratively, applying at each step the *consequence operator* T_q. Many redundant derivations are computed, leading to practical inefficiency. The *semi-naïve* evaluation addresses this problem by considering only facts derived using at least one new fact found at the previous step. Note, however, that while many new facts can be found at one step of the semi-naïve evaluation, only one is to be added by the Best-First method to respect the $\leqslant$-minimality ordering of added facts.

Algorithm 1 Basic semi-naïve version of Best-First method for Datalog provenance

Require: Datalog query q, EDB D with provenance indications over a 0-closed totally-ordered semiring S.
Ensure: full Datalog provenance for the IDB of q.

function Relax($r_0(\vec{x_0})$, S)
  for each instantiation of a rule
    $$r(\vec{x}) \leftarrow r_1(\vec{x_1}), \ldots, r_m(\vec{x_m}), \ldots, r_n(\vec{x_n})$$
  where $r_i(\vec{x_i}) \in D \cup S \cup \{r_0(\vec{x_0})\}$, $1 \leqslant i < m$, $r_m(\vec{x_m}) = r_0(\vec{x_0})$ and $r_i(\vec{x_i}) \in D \cup S$, $m < i \leqslant n$ **do**
    $$v(r(\vec{x})) \quad \oplus\!= \bigotimes_{1 \leqslant i \leqslant n} r_i(\vec{x_i})$$

$I \leftarrow \emptyset$
Let v be the function that maps all facts of D to their annotation in S and all potential facts of the intensional schema of q to $\bar{0}$
for each intensional atom $r(\vec{x}) \notin I$ **do**
  for each instantiation of a rule $r(\vec{x}) \leftarrow r_1(\vec{x_1}), \ldots, r_n(\vec{x_n})$ with $r_i(\vec{x_i}) \in D$ **do**
    $$v(r(\vec{x})) \quad \oplus\!= \bigotimes_{1 \leqslant i \leqslant n} r_i(\vec{x_i})$$
while $\min_{v \setminus I} r(\vec{x}) \neq \bar{0}$ **do**
  Add such minimal $r(\vec{x})$ to I
  Relax($r(\vec{x})$, $I \setminus r(\vec{x})$)
return v

This algorithm starts by initializing the priority queue with IDB facts that are derivable from EDB facts. Then, at each step, the minimum valued-fact is added, and only derivations using this new fact are computed to update the value of the facts in I. This algorithm stops whenever: 1. the maximal value is reached for a candidate fact, or 2. the list is empty - the minimal value of the list is by default the maximal value of the semiring.

Theorem 14. *Algorithm 1 computes the full Datalog provenance for 0-closed totally-ordered semirings.*

Precedence graph. The structure of the Datalog program can be analysed to provide clues about the predicates to focus on. Following [1], we introduce the notion of *precedence graph* G_P of a Datalog program P. The nodes are the IDB predicates and the edges are pairs of IDB predicates (R, R') where R' occurs at the head of a rule of P with R belonging to the tail. P is a *recursive* program if G_P has a directed cycle. Two predicates R and R' are mutually recursive if $R = R'$ or R and R' participate in the same cycle of G_P. This defines equivalence classes. Following a topological ordering on the equivalence classes, Algorithm 1 is sequentially applied to compute the IDB predicates in the current equivalence class, considering previous equivalence classes as EDB predicates. Soufflé natively supports this optimization.

Generalization to distributive lattices. In [19], a new algorithm was introduced for single-source provenance in graph databases over 0-closed multiplicatively idempotent semirings (equivalent to distributive lattices). That method is relevant for semirings that are 0-closed but for which Dijkstra's algorithm is not directly applicable as the semiring is not totally ordered. A similar gap also appears when we consider provenance over Datalog queries (see Section 6). Thus, we show how to apply the method from [19] for computing provenance for Datalog queries over distributive lattices.

Algorithm 2 Generalized Best-first method for Datalog provenance

Require: q a Datalog query with provenance indication over a 0-closed multiplicatively idempotent semiring S.
Ensure: full Datalog provenance for the IDB of q.
1: **for** each EDB fact $R(\vec{x})$**:** Decompose($R(\vec{x})$)
2: **for** each dimension i **:** $v_i \leftarrow$ Best-first(q, i)
3: **return** Recompose$(v_1, \ldots, v_n)$

We provide a brief review of the key ideas presented in [19]. Any element of a distributive lattice is decomposable into a product of *join-irreducible* elements of the lattice, and there exists an embedding of the distributive lattice into a chain decomposition of its join-irreducible elements. This ensures we can 1) work on a totally ordered environment and apply algorithms that require total ordering over the elements, 2) independently compute partial provenance annotations for each dimension to form the final provenance annotation. Given m the number of dimensions in the decomposition, our solution (described in Algorithm 2) performs m launches of the Best-First method and thus, roughly has a cost increased by a factor m.

5 IMPLEMENTATION AND EXPERIMENTS

In numerous application domains, Datalog is used as a *domain specific language* (DSL) to express logical specifications for static program analysis. A formal specification, written as a *declarative* Datalog program is usually translated into an efficient *imperative* implementation by a *synthesizer*. This process simplifies the development of program analysis compared to hand-crafted solutions (highly optimized C++ applications specialized in enforcing a fixed set of specifications). Soufflé [11, 20] has been introduced to provide efficient synthesis of Datalog specifications to executable C++ programs, competing with state-of-the-art handcrafted code for program analysis. The inner workings of Soufflé were of interest to our work; the algorithm implementations are similar to the evaluation strategy followed by the Best-First method we introduced here. We present a brief overview of the architecture of Soufflé and discuss how we extended it.

Architecture and implementation. Following what is described in [11], an input datalog program q goes through a staged specialization hierarchy. After parsing, the first stage of Soufflé specializes the semi-naïve evaluation strategy applied to q, yielding a relational algebra machine program (RAM). Such a program consists in basic relational algebra operations enriched with I/O operators and fix-point computations. As a final step, the RAM program is finally either interpreted or compiled into an executable. For this work, we have only used the interpreter. Our code was inserted in two different stages of Soufflé: a new *translation strategy* from the parsed program to the RAM program, a *priority queue*, replacing the code in charge of adding at run-time the tuples to the relations.

Algorithm 3 Input Datalog program computing the transitive closure (Soufflé syntax)

```
1: .decl edge(s:number, t:number[, @prov:semiring value])
2: .decl path(s:number, t:number[, @prov:semiring value])
3: .input edge .output path
4: path(x, y) :- edge(x, y).
5: path(x, y) :- path(x, z), edge(z, y).
```

Algorithm 4 Corresponding Soufflé RAM program for Algorithm 3

```
1: if ¬(edge = ∅) then
2:     for t0 in edge do
3:         add (t0.0, t0.1) in path
4:         add (t0.0, t0.1) in δpath
5: loop
6:     if ¬(δpath = ∅) ∧ ¬(edge = ∅) then
7:         for t0 in δpath do
8:             for t1 in edge on index t1.0 = t0.1 do
9:                 if ¬(t0.0, t0.1) ∈ path then
10:                    add (t0.0, t0.1) in path'
11:    if path' = ∅ then exit
12:    for t0 in path': add (t0.0, t0.1) in path
13:    swap δpath with path'
14:    clear path
```

We showcase the result of our translation strategy in Algorithms 4 and 5 for a Datalog query computing the transitive closure of a graph; this program is given in Algorithm 3 in its Soufflé syntax. Algorithm 4 presents the corresponding Soufflé RAM program resulting from applying the semi-naïve evaluation strategy and Algorithm 5 our modification to the RAM program to provide provenance annotation via the Best-First strategy and use the priority queue pq for provenance computation. The $\perp$ notation corresponds to a wildcard. Importantly, modifying directly at the RAM level of Soufflé allows us to benefit of all implemented optimizations.

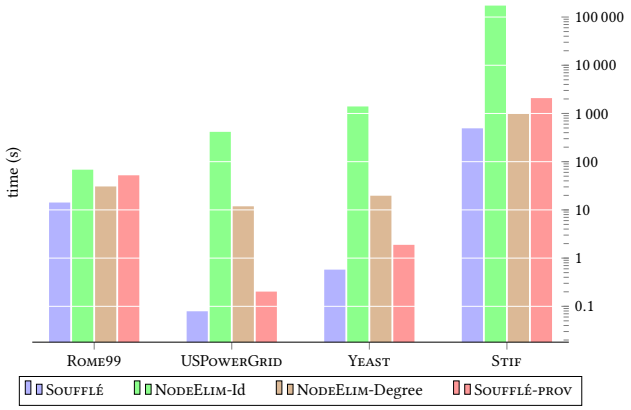

Figure 2: Comparison between algorithms for all-pairs shortest-distances (tropical semiring). Values greater than 100 000 s are timeouts.

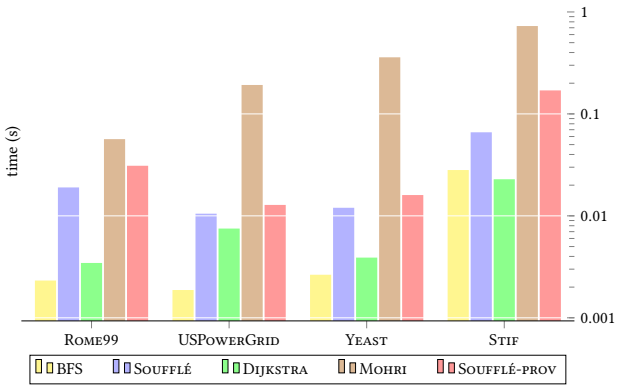

Figure 3: Comparison between algorithms for single-source shortest-distances (tropical semiring).

Table 1: Computation times (in seconds), provenance overhead ratio, size of the output DB (million tuples), and throughput (million tuples/second), for a selection of graph patterns.

	Soufflé				Soufflé-prov				Ratio				Output DB size				Throughput			
Dataset	r	p_1	p_2	p_3	r	p_1	p_2	p_3	r	p_1	p_2	p_3	r	p_1	p_2	p_3	r	p_1	p_2	p_3
Rome99	14.2	.068	6.45	.556	52.2	.134	20.1	1.82	3.68	1.97	3.11	3.26	1.12	.005	16.5	1.31	.792	.047	.820	.718
PowerGrid	.079	.011	.028	.021	.202	.019	.050	.035	2.56	1.45	1.78	1.67	.044	0	.006	.004	.556	n/a	.130	.119
Yeast	.577	.131	7.65	1.30	1.88	.264	25.2	3.73	3.26	2.02	3.56	2.87	.487	ϵ	19.9	3.02	.844	n/a	.789	.808
Stif	491	50.6	oom	oom	2081	141	oom	oom	4.24	2.79	n/a	n/a	313	.068	n/a	n/a	.151	ϵ	n/a	n/a

Algorithm 5 Modification of RAM program of Algorithm 4 to implement Best-First strategy

```
1:  if ¬(edge = ∅) then
2:      for t0 in edge: update (t0.0, t0.1, t0.prov) in path
3:      for t0 in path: add (t0.0, t0.1, t0.prov) in δpath
4:  loop
5:      if ¬(δpath = ∅) ∧ ¬(edge = ∅) then
6:          for t0 in δpath do
7:              for t1 in edge on index t1.0 = t0.1 do
8:                  if ¬(t0.0, t1.1, ⊥) ∈ path then
9:                      update (t0.0, t0.1, t0.prov ⊗ t1.prov) in pq
10:     clear δpath
11:     if pq is empty then exit
12:     add pq.top() in pq.top().relation and in pq.top().δrelation
```

Experiments. Our implementation was tested on an Intel Xeon E5-2650 computer with 176 GB of RAM. The source code will be made available once anonymity requirements are removed.

To translate querying over graphs into Datalog query evaluation, the graph structure has been encoded into an EDB with one binary predicate *edge* encoding the edges, and with edge notations depending on the provenance semiring we chose. We run the *transitive closure* Datalog program outlined in Algorithm 3. We use the same datasets as those used in [19], see [16, 19] for their description and where to download them.

We provide in Figure 2 a comparison between the Best-First method introduced here (Soufflé-prov), the plain Soufflé without provenance computation, and a previous provenance computation algorithm from [19] computing all-pairs shortest-distances over graph databases (the NodeElimination algorithm, with a choice of node to eliminate based on its id or its degree), in the tropical semiring. Similarly, in Figure 3, we compare with previous solutions for single-source shortest-distances, in the same semiring, in particular the adaptation of the Dijkstra algorithm of [19], and, for comparison purposes, a bread-first-search (Bfs) algorithm that simply navigates the graph from the source node but does not compute provenance.

The main focus of this work was to provide an effective Datalog based solution for all-pairs provenance in graph databases. For the all-pairs problem, depending on the dataset, (see, e.g., Yeast), Soufflé-Prov is significantly faster than the previous best known algorithm, NodeElimination. Unsurprisingly, BFS and Dijkstra perform respectively better than Soufflé and Soufflé-prov in the single-source context. What favors both graph algorithms strongly is the fact that they reduce redundant computation: the algorithms abort whenever the target vertex has been reached. Soufflé-prov performs between 1 and 2 orders of magnitude faster than Mohri [17] – an algorithm designed for single-source provenance on k-closed semirings. This fact highlights the potential of adapting the best-first method to also handle k-closed semirings.

Previous work [19] addressing provenance computation for graph databases was restricted to RPQs. We now turn to evaluating this approach for more intricate graph patterns. Patterns considered are

combinations and/or unions of RPQs. The output is moreover not restricted to pairs, but can be of any arbitrary arity. For instance, this allows retrieval of intermediate nodes on a path when computing graph reachability. On the same datasets, we label their edges with two distinct labels, a and b, in an uniform random manner. After this process, some edges have disappeared (neither labeled with a nor b), some appear two times (labeled with a and b, with different weights), or are only associated to one label. The final size of the modified datasets did not change significantly. Table 1 provides a summary of the experiments we conducted over three distinct patterns p_1, p_2, and p_3, described in the following. Pattern $p_1(x, y, z)$:- $R_a(x, y), R_{b^+}(y, z), R_a(z, x)$ selects triplets of vertices that are triangles in the graph, with one side being a b path of arbitrary length. Pattern $p_2(w, x, y, z)$:- $R_{a^+}(w, x), R_{b^+}(x, y), R_{a^+}(y, z)$ selects quadruplets of vertices with two hops, and pattern $p_3(w, x, y, z)$:- $R_{a^+}(w, x), R_b(x, y), R_{a^+}(y, z)$ is a slight variation of p_2. We perform the same experiments as for the reachability queries before, and we indicate the ratio between computation time with or without provenance tracking. The results show that the overhead induced by our provenance approach stays within a constant factor, roughly between 2 and 4, depending on the dataset and pattern. Finally, as observed in the Yeast dataset, the chosen pattern can strongly impact the output DB size: almost negligible having around 100 tuples for pattern p_1, or extremely large having 19 million tuples for pattern p_2. In order to provide a meaningful comparison, we also measure the *throughput*, consisting in the average number of output tuples processed per second. For smaller output DB sizes, this measure is less relevant, as the fixed costs of running Soufflé-prov dominate the overall running time. Overall, these results are promising, as our method has the potential to process 1M tuples per second when the query output is large.

6 RELATED WORK

With respect to Datalog provenance, it has been shown in [6] that, for a Datalog program having n candidate IDB tuples, a circuit for representing Datalog provenance in the semiring $\mathbf{Sorp}(X)$ (the most general absorptive semiring) only needs $n + 1$ layers. For binary relations, e.g., representing the edge relation of a graph, this construction is at least quadratic in the number of vertices, thus not practically applicable for the graphs we analyzed in our experiments. Similarly, in [7], absorptive semirings (i.e., 0-closed semirings) have the property that derivation trees of size $\geqslant n$ are "pumpable" (they do not contribute to the final result). A concrete implementation [8] computes the provenance for commutative and idempotent semirings using n Newton iterations.

Fairly recently, [12] introduced POPS (*Partially Ordered, Pre-Semiring*), a structure decoupling the order on which the fixed-point is computed from the semiring structure. Complex and recursive computations over vectors, matrices, tensors are now expressible using this framework. The study also generalized the semi-naïve method from plain Datalog evaluation to idempotent semirings (aka dioids). In comparison, our method is restricted to semirings that are totally ordered (a subclass of distributive dioids[2]), leveraging the invariant that once a fact is first labeled with a provenance value, we are certain it is the correct one.

In cases where keeping the *full* provenance of a program (*how*-provenance) is still prohibitively large, [4, 5] propose to select only a relevant subset of such trees using *selection criteria* based on tree patterns and ranking over rules and facts occurring in the derivation. First, given a Datalog program P and a pattern q, an *offline* instrumentation is performed, leading to an *instrumented program* P_q. Then, given any database D, an efficient algorithm can be used to retrieve only the top-k best derivation trees for $P_q(D)$. The top-1 algorithm of the study is closely related to our solution, but does not mention the use of a priority queue nor does it take into account the optimization provided by the semi-naïve evaluation strategy we describe in Section 4.

Our solution can be seen as a hybrid of the ideas introduced in [12] and [5]. We generalize the semi-naïve evaluation to a specific class of semirings in order to achieve a more efficient algorithm, one that can be used in practical real-world scenarios.

7 CONCLUSION

In this work, we developed a novel method for Datalog provenance computation based on the link between dynamic programming over hypergraphs and the proof structure of provenance of Datalog programs. We introduced Knuth's algorithm for computing the provenance, and optimized it for practical use. We showed its feasibility by providing an implementation on top of Soufflé and tested it on several graph databases and matching patterns.

We proved in this paper that optimization methods for graph provenance naturally extend to Datalog provenance, for certain classes of semirings playing a major role in practical applications. The previous work targeting provenance computations for graph databases in the framework of [18] and [19] only considered RPQs, which are a strict subset of the expressive capabilities of Datalog. We thus have extended the supported set of queries for provenance-aware interrogation of graph databases, maintaining the practical efficiency of the approach. The theoretical complexity of the introduced method is not fully understood yet; we plan to address this in future work.

The internals of Soufflé, targeting the inflationary computation of the fixed-point operator lack support for updating tuples. We conjecture we could mitigate the overhead induced by provenance computations within Soufflé-prov by adding primitives in their data structures. What remains to be established is to what extent these data structures [11] could be extended to handle updates, without reducing the efficiency of Soufflé's current set of operators.

ACKNOWLEDGMENTS

This work has been funded by the French government under management of Agence Nationale de la Recherche as part of the "Investissements d'avenir" program, reference ANR-19-P3IA-0001 (PRAIRIE 3IA Institute).

REFERENCES

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] Distributive dioids are POPS structures over a distributive lattice being the natural order of the dioid.

[2] Pablo Barceló. Querying graph databases. In *PODS*, pages 175–188, New York, 2013. ACM.
[3] James Cheney, Laura Chiticariu, and Wang Chiew Tan. Provenance in databases: Why, how, and where. *Found. Trends Databases*, 1(4):379–474, 2009.
[4] Daniel Deutch, Amir Gilad, and Yuval Moskovitch. selP: selective tracking and presentation of data provenance. In *ICDE*, pages 1484–1487, 2015.
[5] Daniel Deutch, Amir Gilad, and Yuval Moskovitch. Efficient provenance tracking for datalog using top-k queries. *The VLDB Journal*, 27:245–269, 2018.
[6] Daniel Deutch, Tova Milo, Sudeepa Roy, and Val Tannen. Circuits for Datalog Provenance. In *ICDT*, pages 201–212, 2014.
[7] Javier Esparza and Michael Luttenberger. Solving fixed-point equations by derivation tree analysis. In *CALCO*, pages 19–35, 2011.
[8] Javier Esparza, Michael Luttenberger, and Maximilian Schlund. Fpsolve: A generic solver for fixpoint equations over semirings. In *CIAA*, pages 1–15, 2014.
[9] Todd J. Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *PODS*, 2007.
[10] Liang Huang. Advanced dynamic programming in semiring and hypergraph frameworks. In *COLING*, pages 1–18, 2008.
[11] Herbert Jordan, Bernhard Scholz, and Pavle Subotić. Soufflé: On synthesis of program analyzers. In *CAV*, pages 422–430, 2016.
[12] Mahmoud Abo Khamis, Hung Q. Ngo, Reinhard Pichler, Dan Suciu, and Yisu Remy Wang. Convergence of datalog over (pre-) semirings. *CoRR*, abs/2105.14435, 2021.
[13] Donald E. Knuth. A generalization of Dijkstra's algorithm. *Information Processing Letters*, 6(1), 1977.
[14] Daniel Krob. Monoides et semi-anneaux complets. In *Semigroup Forum*, volume 36, pages 323–339. Springer, 1987.
[15] Werner Kuich. Semirings and formal power series: Their relevance to formal languages and automata. In *Handbook of Formal Languages*, volume 1, chapter 9, pages 609–677. Springer, 1997.
[16] Silviu Maniu, Pierre Senellart, and Suraj Jog. An Experimental Study of the Treewidth of Real-World Graph Data. In *ICDT*, pages 12:1–12:18, 2019.
[17] Mehryar Mohri. Semiring frameworks and algorithms for shortest-distance problems. *J. Autom. Lang. Comb.*, 7(3):321–350, 2002.
[18] Yann Ramusat, Silviu Maniu, and Pierre Senellart. Semiring provenance over graph databases. In *TaPP*, 2018.
[19] Yann Ramusat, Silviu Maniu, and Pierre Senellart. Provenance-based algorithms for rich queries over graph databases. In *EDBT*, pages 73–84, 2021.
[20] Bernhard Scholz, Herbert Jordan, Pavle Subotić, and Till Westmann. On fast large-scale program analysis in datalog. In *CC*, page 196–206, 2016.
[21] Pierre Senellart. Provenance in databases: Principles and applications. In *Reasoning Web*, pages 104–109, 2019.

A PROOFS FOR SECTION 2 (BACKGROUND)

Lemma 2. *Let S be an idempotent semiring and $\leqslant$ the natural order over S. Then S is* superior *with respect to $\leqslant$ if and only if S is* 0*-closed.*

Proof. First assume S superior with respect to $\leqslant$. Then for any a, $\bar{1} \leqslant \bar{1} \otimes a = a$, which means that $\bar{1} + a = \bar{1}$, i.e., S is 0-closed.

Now assume S 0-closed. Since $a \oplus a \otimes b = a \otimes (\bar{1} \oplus b) = a$, we have: $a \leqslant a \otimes b$, and similarly for $b \leqslant a \otimes b$. Thus S is superior with respect to $\leqslant$. □

B PROOFS FOR SECTION 3 (DATALOG PROVENANCE AND DYNAMIC PROGRAMMING OVER HYPERGRAPHS)

Lemma 11. *For any Datalog query q and grounding of an atom v of q, there is a bijection between $\mathcal{D}_{Hq}(v)$ and $\{\tau \mid \tau \text{ yields } v\}$.*

Proof. By definition of H_q each instantiation of a rule corresponds to a unique hyperedge. Then, we can inductively construct for a given derivation D its associated (unique) Datalog proof tree τ_D:

- If $|D| = 1$, then v is a source vertex and thus an extensional tuple, we get the empty proof.
- If $|D| \geqslant 1$, then there exists $e \in \mathrm{BS}(v)$ where $|e| \geqslant 0$ and D_i a derivation of $T_i(e)$ for $1 \leqslant i \leqslant |e|$, where $D = \langle e, D_1 \cdots D_{|e|} \rangle$. By definition, this hyperedge corresponds to the grounding of a rule $t(\vec{x}) \leftarrow r_1(\vec{x_1}), \ldots, r_n(\vec{x_n})$. By induction, for $1 \leqslant i \leqslant |e|$, τ_{D_i} is the corresponding proof of the derivation D_i. Then by composition we obtain τ_D the proof for D.

□

Lemma 12. *For any Datalog query q and grounding of an atom v of q, for any derivation D of v in H_q we have*

$$w(D) = \bigotimes_{t' \in \mathit{leaves}(\tau_D)} \mathrm{prov}_R^q(t')$$

where τ_D is the proof tree corresponding to D in the bijection given by Lemma 11.

Proof. By induction on the size of the derivation D:

- If $|D| = 1$ then, there exists a nullary edge $e \in E_q$ with $h(e) = v$ and $w(D) = f_v = \mathrm{prov}_R^q(r(\vec{x})) = \prod_{t' \in \mathit{leaves}(\tau_D)} \mathrm{prov}_R^q(t')$.
- If $|D| \geqslant 1$ then there exists $e \in E_q$ and D is of the form $\langle e, D_1 \cdots D_{|e|} \rangle$ with D_i a derivation of $T_i(e)$ for $1 \leqslant i \leqslant |e|$. We have $w(D) = f_e(w(D_1), \ldots, w(D_{|e|}))$ and by definition of $f_e = \otimes$ and by IHP $w(D) = \bigotimes_{t' \in \mathit{leaves}(\tau_D)} \mathrm{prov}_R^q(t')$.

□

Theorem 13. *Let t be a tuple of a Datalog program q with output predicate G and H_q its hypergraph representation, then* $\mathrm{prov}_G^q(t) = \delta_{H_q}(G(t))$.

Proof.

$$\begin{aligned}
\delta_{H_q}(t) &= \bigoplus_{D \in \mathcal{D}_{Hq}(G(t))} w(D) && \text{and by Lemma 12,} \\
&= \bigoplus_{D \in \mathcal{D}_{Hq}(G(t))} \left(\bigotimes_{t' \in \mathit{leaves}(\tau_D)} \mathrm{prov}_R^q(t') \right) && \text{and by Lemma 11,} \\
&= \bigoplus_{\tau \text{ yields } t} \left(\bigotimes_{t' \in \mathit{leaves}(\tau)} \mathrm{prov}_R^q(t') \right) && = \quad \mathrm{prov}_T^q(t).
\end{aligned}$$

□

C PROOFS FOR SECTION 4 (BEST-FIRST METHOD)

Theorem 14. *Algorithm 1 computes the full Datalog provenance for* 0*-closed totally-ordered semirings.*

Proof. We show the algorithm verifies the following invariant: whenever a tuple is added to I in Line 11, it has optimal value. This implies that I is populated in increasing order: each new derivation computed in the Relax() procedure only updates the priority queues with values greater than the value of the tuple relaxed (by superiority of $\otimes$).

Assume by contradiction that some output tuples are not correctly labeled and take such a minimal tuple $v = r(\vec{x})$. At the moment where v is extracted with value n let us consider an optimal derivation path of v that leads to the optimum value $opt < n$. By superiority each tuple occurring in the tail of the rule has value less than opt. Thus a tuple occurring in the tail is either wrong-valued or not present in I at the moment where v is found. In both cases and because tuples are added to I in increasing order we obtain a new minimal tuple incorrectly labeled by the algorithm, contradicting the hypothesis. □