Efficient Provenance-Aware Querying of Graph Databases with Datalog

Yann Ramusat Silviu Maniu *Pierre Senellart*

Grades-NDA 2022, 12 June 2022

Provenance Annotations

Provenance annotations provide additional information within a database to gain more information about query results.

These annotations are propagated to query results and can be used for example to:

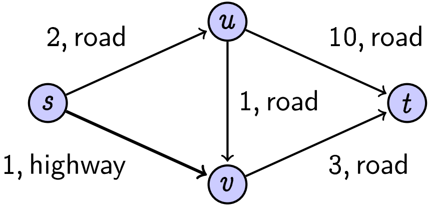
- determine how the result has been computed;
- understand how it would reacts to slight changes in the initial database;
- perform computations alongside query evaluation.

Semiring-Based Provenance

A strong mathematical foundation is to choose provenance annotations to be elements of a semiring (Green et al., 2007).

Semirings are a well-suited model for operations (e.g., choices and sequences) carried along in computations.

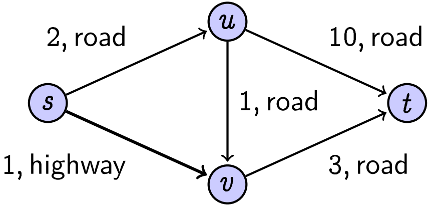
Working Example (Tropical Semiring)



These integers represent time to move between two vertices.

What is the minimum travel time between s and t?

Working Example (Tropical Semiring)

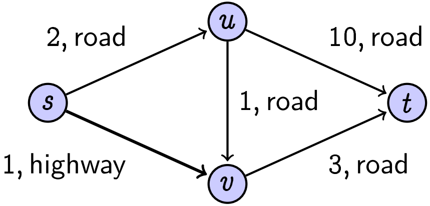


These integers represent time to move between two vertices.

What is the minimum travel time between s and t?

And now, if we only consider paths avoiding highways?

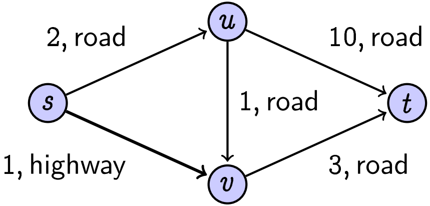
Working Example (Counting Semiring)



These integers represent number of paths between two vertices.

What is the total number of paths between s and t?

Working Example (Counting Semiring)

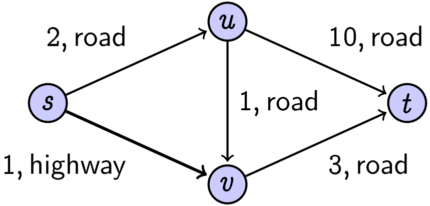


These integers represent number of paths between two vertices.

What is the total number of paths between s and t?

And now, if we only consider paths avoiding highways?

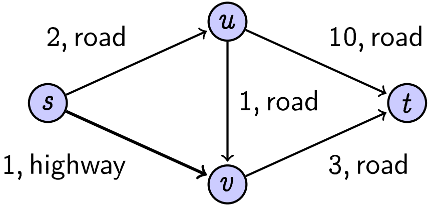
Working Example (Top-2 Semiring)



These integers represent time to move between two vertices.

What are the best two travel times between s and t?

Working Example (Top-2 Semiring)

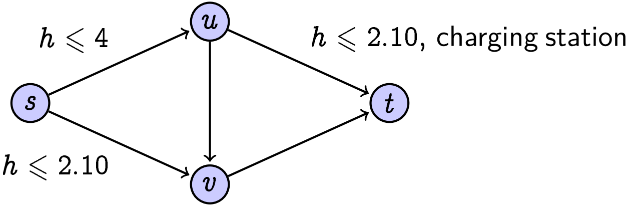


These integers represent time to move between two vertices.

What are the best two travel times between s and t?

And now, if we only consider paths avoiding highways?

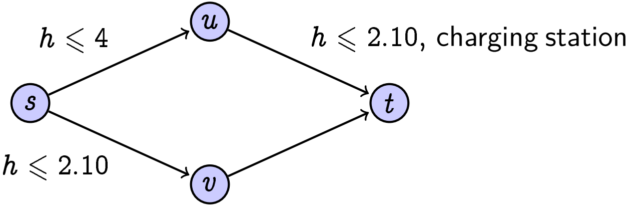
Working Example (k-feature Semiring)



There exists a path from s to t going through a charging station.

There exists another one allowing 3m high vehicles to reach t.

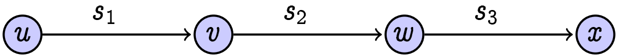
Working Example (k-feature Semiring)



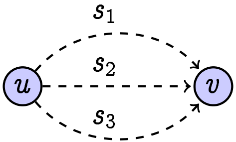
There exists a path from s to t going through a charging station.

There does not exist one permitting 3m high vehicles to reach t.

Algebraic Foundations – Operators $\oplus$ and $\otimes$



$\otimes$-associativity: $s_1 \otimes s_2 \otimes s_3 := (s_1 \otimes s_2) \otimes s_3 = s_1 \otimes (s_2 \otimes s_3)$

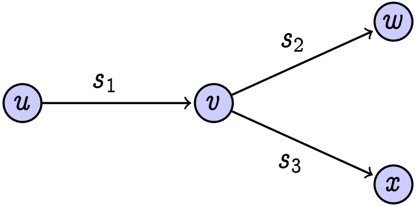


$\oplus$-commutativity: $s_1 \oplus s_2 = s_2 \oplus s_1$

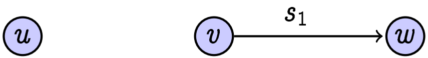
$\oplus$-associativity: $s_1 \oplus s_2 \oplus s_3 := (s_1 \oplus s_2) \oplus s_3 = s_1 \oplus (s_2 \oplus s_3)$

neutral $\oplus$ element: $\oplus_\emptyset := \bar{0}$ neutral $\otimes$ element: $\otimes_\emptyset := \bar{1}$

Algebraic Foundations – Mixing both Operators

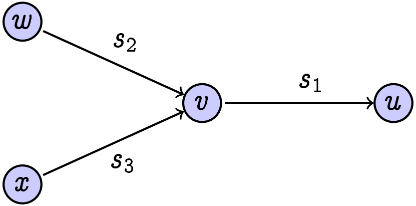


$\otimes$ distributivity over $\oplus$: $s_1 \otimes (s_2 \oplus s_3) = (s_1 \otimes s_2) \oplus (s_1 \otimes s_3)$

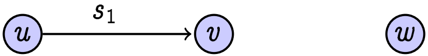


$\bar{0}$ annihilates $\otimes$: $\bar{0} \otimes s_1 = \bar{0}$

Algebraic Foundations – Mixing both Operators



$\otimes$ distributivity over $\oplus$: $(s_2 \oplus s_3) \otimes s_1 = (s_2 \otimes s_1) \oplus (s_3 \otimes s_1)$



$\bar{0}$ annihilates $\otimes$: $s_1 \otimes \bar{0} = \bar{0}$

Semirings – Basic Properties

Some semirings may satisfy additional properties:

- commutativity: for all $a, b \in S$, $a \otimes b = b \otimes a$;
- 0-closed, bounded: for all $a \in S$, $1 \oplus a = 1$;
- pre-order: $a \sqsubseteq_S b := \exists h \in S, a \oplus h = b$:
  - smallest element: for all $a \in S$, $\bar{0} \sqsubseteq_S a$;
  - monotonicity: $a \sqsubseteq_S b \implies a \oplus c \sqsubseteq_S b \oplus c \wedge a \otimes c \sqsubseteq_S b \otimes c$.
- when $\sqsubseteq_S$ is a partial order it is called the natural order $\leqslant_S$:
  - 0-closed implies $\leqslant_S$ is a partial order;
  - a semiring need not be 0-closed to be naturally ordered.

Semirings – Examples

- Tropical semiring $(\min, +)$:
  $\rightarrow$ 0-closed, commutative, $\leqslant_S = \text{rev}(\leqslant_{\mathbb{N}})$ is total.
- Counting semiring $(+, \times)$:
  $\rightarrow$ commutative, $\leqslant_S = \leqslant_{\mathbb{N}}$ is total.
- Top-k (distinct) semiring $(\min^k, +^k)$:
  $\rightarrow$ (0-closed), commutative, $\leqslant_S$ is partial.
- k-feature semiring $(\min^k, \max^k)$:
  $\rightarrow$ 0-closed, commutative, $\leqslant_S$ is a lattice order.

Semirings – Examples

- Tropical semiring $(\min, +)$:
  $\rightarrow$ 0-closed, commutative, $\leqslant_S = \text{rev}(\leqslant_{\mathbb{N}})$ is total.
- Counting semiring $(+, \times)$:
  $\rightarrow$ commutative, $\leqslant_S = \leqslant_{\mathbb{N}}$ is total.
- Top-k (distinct) semiring $(\min^k, +^k)$:
  $\rightarrow$ (0-closed), commutative, $\leqslant_S$ is partial.
- k-feature semiring $(\min^k, \max^k)$:
  $\rightarrow$ 0-closed, commutative, $\leqslant_S$ is a lattice order.

Dijkstra for provenance (Ramusat et al., 2021)

- When the semiring is 0-closed and the natural order is total, possible to compute the provenance using Dijkstra algorithm: maintain a priority queue of nodes encountered but not yet processed, ordered according to the natural order of the provenance expression computed so far for that node.
- When the semiring is 0-closed and the order is not necessarily total but a lattice order of finite dimension, possible to apply Dijkstra on each dimension of the lattice.

Contents

Definition of the Model

Definition (Graph database)

A *graph database* G over Σ is a pair (V, E). V finite set of node ids.
An edge in G is a triple $(v, a, v') \in V \times \Sigma \times V$, whose interpretation is an a-labeled edge from v to v' in G.

Definition (Graph database with provenance indication)

A *graph database with provenance indication* (V, E, w) over S is a graph database (V, E) together with a *weight function*, $w\colon E \to S$ for $(S, \oplus, \otimes, \bar{0}, \bar{1})$ a semiring.

Weighted Sets of Paths

- Extend the weight function w to paths: $w[\pi] := \bigotimes_{i=1}^{k} w[e_i]$.
- And further to any finite set of paths:

$$w[\bigcup_{i=1}^{n} \pi_i] := \bigoplus_{i=1}^{n} w[\pi_i].$$

- Denote by $\rho(e)$ the label of an edge $e \in E$.
- Extend labels to paths, $\rho(\pi) \in \Sigma^*$:

$$\rho(\pi) = \rho(e_1)\rho(e_2)\cdots\rho(e_{k-1})\rho(e_k).$$

Contents

Motivations

We leverage these three facts:

- Datalog is a very expressive framework for expressing queries;
- very rich literature around Datalog and Datalog provenance;
- some practical systems are built on top of Datalog;

→ to obtain new (and better!) effective solutions to practical scenarios (i.e., real transportation networks over large areas);

→ to process queries that go beyond the simple class of RPQs.

Overview of SOUFFLÉ (Scholz et al., 2016)

SOUFFLÉ is a *logic programming language* based on Datalog.

Designed to perform efficient synthesis of static program analysis specifications, employing Datalog as a domain specific language.

SOUFFLÉ's relevant features for us:

- competes with hand-written specifications for static program analysis;
- does not restrict to a specific target application;
- comes along with its own optimized data structures;
- possesses an *(informational)* provenance evaluation strategy for debugging (Zhao et al., 2020).

Best-First Method

We adapt the generalization of Dijkstra's algorithm to the *grammar problem* due to Knuth (1977).

This permits to compute Datalog provenance over 0-closed semirings having a total natural order.

Dijkstra is a subcase, corresponding to right (or left) linear Datalog programs.

Extending the Semi-Naïve Evaluation Strategy

But... how do we get an efficient implementation?

- consider each instantiation of a rule only once, when all the premises are provenance annotated:
  → update the tentative provenance for the head, in the priority queue,
  → if the head is already in the IDB, it has a better annotation!
- only consider mutually recursive predicates to mitigate the load of the priority queue.

We basically apply the semi-naïve evaluation strategy, with a small twist!

Introducing Soufflé-Prov

We implement the best-first method, adapting Soufflé's semi-naïve evaluation strategy powered by its efficient data structures, and set of optimizations.

→ Soufflé-Prov is to Soufflé what ProvSQL (Senellart et al., 2018) is to PostgreSQL.

Key points:

- We do not break any of Soufflé's optimizations!
- The lattice-theoretic approach stays applicable in this context.

Datalog Program for Transitive Closure

Algorithm 1 Transitive Closure (Soufflé syntax)

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

Corresponding Soufflé RAM Program

Algorithm 2 RAM Program for Transitive Closure

```
if ¬(edge = ∅) then
    for t0 in edge:  add (t0.0, t0.1) in path
    for t0 in path:  add (t0.0, t0.1) in δpath
loop
    if ¬(δpath = ∅) ∧ ¬(edge = ∅) then
        for t0 in δpath do
            for t1 in edge on index t1.0 = t0.1 do
                if ¬(t0.0, t0.1) ∈ path then
                    add (t0.0, t0.1) in path'
    if path' = ∅ then exit
    for t0 in path':  add (t0.0, t0.1) in path
    swap δpath with path'
    clear path
```

Corresponding Soufflé-Prov RAM Program

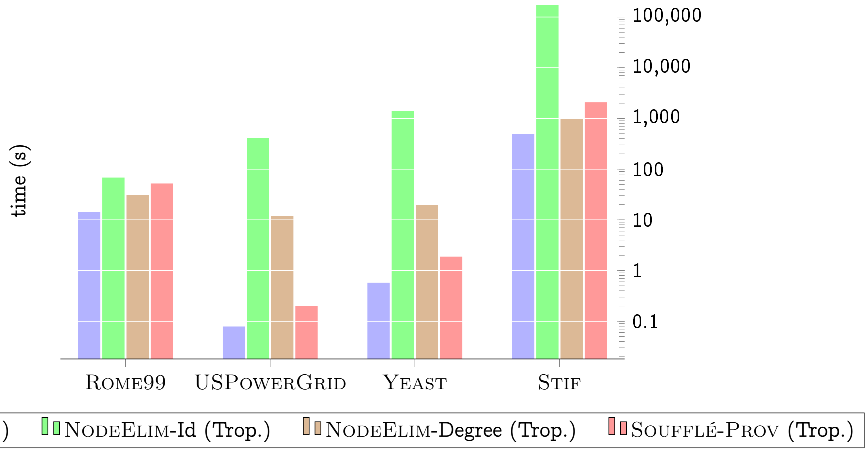
Algorithm 3 RAM Program for Provenance-Aware Transitive Closure

```
if ¬(edge = ∅) then
    for t0 in edge:  update* (t0.0, t0.1, t0.prov) in path
    for t0 in path:  add (t0.0, t0.1, t0.prov) in δpath
loop
    if ¬(δpath = ∅) ∧ ¬(edge = ∅) then
        for t0 in δpath do
            for t1 in edge on index t1.0 = t0.1 do
                if ¬(t0.0, t1.1, ⊥) ∈ path then
                    update (t0.0, t0.1, t0.prov ⊗ t1.prov) in pq
    clear δpath
    If pq is empty then exit
    add pq.top() in pq.top().relation and in pq.top().δrelation
```

pq may contain tuples from every mutually recursive predicate!

Computing All-Pairs Shortest Distances

Comparison between algorithms for all-pairs shortest distances:



Efficiency for a Selection of Graph Patterns

Patterns:

- $\mathsf{r}(x, y)$:- $\mathsf{path}(x, z)$
- $\mathsf{p}_1(x, y, z)$:- $\mathsf{edge}_\mathsf{a}(x, y), \mathsf{path}_\mathsf{b}(y, z), \mathsf{edge}_\mathsf{a}(z, x)$
- $\mathsf{p}_2(w, x, y, z)$:- $\mathsf{path}_\mathsf{a}(w, x), \mathsf{path}_\mathsf{b}(x, y), \mathsf{path}_\mathsf{a}(y, z)$
- $\mathsf{p}_3(w, x, y, z)$:- $\mathsf{path}_\mathsf{a}(w, x), \mathsf{edge}_\mathsf{b}(x, y), \mathsf{path}_\mathsf{a}(y, z)$

For relevant output DB sizes (containing from 0.5M to 20M tuples):

- SOUFFLÉ-PROV is 2.8 to 3.6 times slower than SOUFFLÉ,
- up-to 1M output tuples processed by seconds.

Contents

In brief

- Efficient computation of provenance of graph databases is possible, for a rich class of queries (Datalog), and with a reasonable overhead
- ... as long as the provenance semiring is 0-closed, and either naturally ordered or a lattice order with low dimension
- Perspectives:
    - Further optimizations, getting as close as possible to the performance of standard Datalog Evaluation
    - Beyond 0-closed semirings: k-closed semirins, locally k-closed semirings, etc.

Bibliography I

Green, T. J., Karvounarakis, G., and Tannen, V. (2007). Provenance semirings. In *PODS*, pages 31–40.

Knuth, D. E. (1977). A generalization of Dijkstra's algorithm. *Information Processing Letters*, 6(1).

Ramusat, Y., Maniu, S., and Senellart, P. (2021). Provenance-Based Algorithms for Rich Queries over Graph Databases. In *EDBT*.

Scholz, B., Jordan, H., Subotić, P., and Westmann, T. (2016). On fast large-scale program analysis in datalog. In *International Conference on Compiler Construction*, page 196–206.

Senellart, P., Jachiet, L., Maniu, S., and Ramusat, Y. (2018). Provsql: Provenance and probability management in postgresql. *Proc. VLDB Endow.*, 11(12):2034–2037.

Bibliography II

Zhao, D., Subotić, P., and Scholz, B. (2020). Debugging large-scale datalog: A scalable provenance evaluation strategy. *ACM TOPLAS*, 42(2).