Challenges in Building a Provenance-Aware Database Management System

Pierre Senellart

Simons Institute, 7 September 2023

Provenance management

- Data management all about query evaluation

Provenance management

- Data management all about query evaluation
- What if we want something more than the query result?
  - Where does the result come from?
  - Why was this result obtained?
  - How was the result produced?
  - What is the probability of the result?
  - How many times was the result obtained?
  - How would the result change if part of the input data was missing?
  - What is the minimal security clearance I need to see the result?
  - What is the most economical way of obtaining the result?
  - How can a result be explained in layman terms?

Provenance management

- Data management all about query evaluation
- What if we want something more than the query result?
  - Where does the result come from?
  - Why was this result obtained?
  - How was the result produced?
  - What is the probability of the result?
  - How many times was the result obtained?
  - How would the result change if part of the input data was missing?
  - What is the minimal security clearance I need to see the result?
  - What is the most economical way of obtaining the result?
  - How can a result be explained in layman terms?
- Provenance management: along with query evaluation, record additional bookkeeping information allowing to answer the questions above

Outline

Data model: annotated relations

- Relational data model: data decomposed into relations, with labeled attributes...

Data model: annotated relations

- Relational data model: data decomposed into relations, with labeled attributes...

name	position	city	classification
John	Director	New York	unclassified
Paul	Janitor	New York	restricted
Dave	Analyst	Paris	confidential
Ellen	Field agent	Berlin	secret
Magdalen	Double agent	Paris	top secret
Nancy	HR director	Paris	restricted
Susan	Analyst	Berlin	secret

Data model: annotated relations

- Relational data model: data decomposed into relations, with labeled attributes...
- ... with an extra provenance annotation for each tuple (think of it first as a tuple id)

name	position	city	classification	prov
John	Director	New York	unclassified	x_1
Paul	Janitor	New York	restricted	x_2
Dave	Analyst	Paris	confidential	x_3
Ellen	Field agent	Berlin	secret	x_4
Magdalen	Double agent	Paris	top secret	x_5
Nancy	HR director	Paris	restricted	x_6
Susan	Analyst	Berlin	secret	x_7

Queries

- A query is an arbitrary function that maps databases over a fixed database schema $\mathcal{D}$ to relations over some relational schema $\mathcal{R}$
- The query does not consider or produce any provenance annotations; we will give semantics for the provenance annotations of the output, based on that of the input
- In practice, one often restricts to specific query languages:
  - Monadic-Second Order logic (MSO)
  - First-Order logic (FO) or the relational algebra, or fragments thereof
  - SQL with aggregate functions
  - etc.

Outline

Boolean provenance [Imieliński and Lipski, 1984]

- $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ finite set of Boolean events
- Provenance annotation: Boolean function over $\mathcal{X}$, i.e., a function of the form: $(\mathcal{X} \to \{\bot, \top\}) \to \{\bot, \top\}$
- Interpretation: possible-world semantics
  - every valuation $\nu : \mathcal{X} \to \{\bot, \top\}$ denotes a possible world of the database
  - the provenance of a tuple on ν evaluates to $\bot$ or $\top$ depending whether this tuple exists in that possible world
  - for example, if every tuple of a database is annotated with the indicator function of a distinct Boolean event, the set of possible worlds is the set of all subdatabases

Example of possible worlds

name	position	city	classification	prov
John	Director	New York	unclassified	x_1
Paul	Janitor	New York	restricted	x_2
Dave	Analyst	Paris	confidential	x_3
Ellen	Field agent	Berlin	secret	x_4
Magdalen	Double agent	Paris	top secret	x_5
Nancy	HR director	Paris	restricted	x_6
Susan	Analyst	Berlin	secret	x_7

$$\nu: \begin{array}{ccccccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \\ \top & \top & \top & \top & \top & \top & \top \end{array}$$

Example of possible worlds

name	position	city	classification	prov
John	Director	New York	unclassified	x_1
Dave	Analyst	Paris	confidential	x_3
Magdalen	Double agent	Paris	top secret	x_5
Susan	Analyst	Berlin	secret	x_7

$$\nu: \begin{array}{ccccccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \\ \top & \bot & \top & \bot & \top & \bot & \top \end{array}$$

Boolean provenance of query results

- $\nu(D)$: the subdatabase of D where all tuples whose provenance annotation evaluates to $\bot$ by ν are removed
- The Boolean provenance $\mathrm{prov}_{q,D}(t)$ of tuple $t \in q(D)$ is the function:

$$\nu \mapsto \begin{cases} \top \text{ if } t \in q(\nu(D)) \\ \bot \text{ otherwise} \end{cases}$$

Example (What cities are in the table?)

name	position	city	classification	prov
John	Director	New York	unclassified	x_1
Paul	Janitor	New York	restricted	x_2
Dave	Analyst	Paris	confidential	x_3
Ellen	Field agent	Berlin	secret	x_4
Magdalen	Double agent	Paris	top secret	x_5
Nancy	HR director	Paris	restricted	x_6
Susan	Analyst	Berlin	secret	x_7

city	prov
New York	$x_1 \vee x_2$
Paris	$x_3 \vee x_5 \vee x_6$
Berlin	$x_4 \vee x_7$

What now?

- How to compute Boolean provenance for practical query languages? What complexity?
- Example application of provenance: probabilistic databases
- How should we represent provenance annotations?
- How can we implement support for provenance management in a relational database management system?

Outline

Commutative semiring $(K, \mathbb{0}, \mathbb{1}, \oplus, \otimes)$

- Set K with distinguished elements $\mathbb{0}$, $\mathbb{1}$
- $\oplus$ associative, commutative operator, with identity $\mathbb{0}_K$:
  - $a \oplus (b \oplus c) = (a \oplus b) \oplus c$
  - $a \oplus b = b \oplus a$
  - $a \oplus \mathbb{0} = \mathbb{0} \oplus a = a$
- $\otimes$ associative, commutative operator, with identity $\mathbb{1}_K$:
  - $a \otimes (b \otimes c) = (a \otimes b) \otimes c$
  - $a \otimes b = b \otimes a$
  - $a \otimes \mathbb{1} = \mathbb{1} \otimes a = a$
- $\otimes$ distributes over $\oplus$:

$$a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$$

- $\mathbb{0}$ is annihilating for $\otimes$:

$$a \otimes \mathbb{0} = \mathbb{0} \otimes a = \mathbb{0}$$

Example semirings

- $(\mathbb{N}, 0, 1, +, \times)$: counting semiring
- $(\{\bot, \top\}, \bot, \top, \vee, \wedge)$: Boolean semiring
- $(\{\mathit{unclassified}, \mathit{restricted}, \mathit{confidential}, \mathit{secret}, \mathit{top\ secret}, \mathit{unavailable}\}, \mathit{unavailable}, \mathit{unclassified}, \min, \max)$: security semiring
- $(\mathbb{N} \cup \{\infty\}, \infty, 0, \min, +)$: tropical semiring
- $(\{\text{Boolean functions over } \mathcal{X}\}, \bot, \top, \vee, \wedge)$: semiring of Boolean functions over $\mathcal{X}$
- $(\mathbb{N}[\mathcal{X}], 0, 1, +, \times)$: semiring of integer-valued polynomials with variables in $\mathcal{X}$ (also called How-semiring or universal semiring, see further)
- $(\mathcal{P}(\mathcal{P}(\mathcal{X})), \emptyset, \{\emptyset\}, \cup, \Cup)$: Why-semiring over $\mathcal{X}$ $(A \Cup B := \{a \cup b \mid a \in A, b \in B\})$

Semiring provenance [Green et al., 2007]

- We fix a semiring $(K, \mathbb{0}, \mathbb{1}, \oplus, \otimes)$
- We assume provenance annotations are in K
- We consider a query q from the positive relational algebra (selection, projection, renaming, cross product, union; joins can be simulated with renaming, cross product, selection, projection)
- We define a semantics for the provenance of a tuple $t \in q(D)$ inductively on the structure of q

Selection, renaming

Provenance annotations of selected tuples are unchanged

Example ($\rho_{\text{name}\to\text{n}}(\sigma_{\text{city}=\text{"New York"}}(R))$)

name	position	city	classification	prov
John	Director	New York	unclassified	x_1
Paul	Janitor	New York	restricted	x_2
Dave	Analyst	Paris	confidential	x_3
Ellen	Field agent	Berlin	secret	x_4
Magdalen	Double agent	Paris	top secret	x_5
Nancy	HR director	Paris	restricted	x_6
Susan	Analyst	Berlin	secret	x_7

n	position	city	classification	prov
John	Director	New York	unclassified	x_1
Paul	Janitor	New York	restricted	x_2

Projection

Provenance annotations of identical, merged, tuples are $\oplus$-ed

Example ($\pi_{\text{city}}(R)$)

name	position	city	classification	prov
John	Director	New York	unclassified	x_1
Paul	Janitor	New York	restricted	x_2
Dave	Analyst	Paris	confidential	x_3
Ellen	Field agent	Berlin	secret	x_4
Magdalen	Double agent	Paris	top secret	x_5
Nancy	HR director	Paris	restricted	x_6
Susan	Analyst	Berlin	secret	x_7

city	prov
New York	$x_1 \oplus x_2$
Paris	$x_3 \oplus x_5 \oplus x_6$
Berlin	$x_4 \oplus x_7$

Union

Provenance annotations of identical, merged, tuples are $\oplus$-ed

Example

$\pi_{\text{city}}(\sigma_{\textit{ends-with}(\text{position},\text{“agent”})}(R)) \cup \pi_{\text{city}}(\sigma_{\text{position}=\text{“Analyst”}}(R))$

name	position	city	classification	prov
John	Director	New York	unclassified	x_1
Paul	Janitor	New York	restricted	x_2
Dave	Analyst	Paris	confidential	x_3
Ellen	Field agent	Berlin	secret	x_4
Magdalen	Double agent	Paris	top secret	x_5
Nancy	HR director	Paris	restricted	x_6
Susan	Analyst	Berlin	secret	x_7

city	prov
Paris	$x_3 \oplus x_5$
Berlin	$x_4 \oplus x_7$

Cross product

Provenance annotations of combined tuples are $\otimes$-ed

Example

$$\pi_{\text{city}}(\sigma_{\textit{ends-with}(\text{position},\text{“agent”})}(R)) \bowtie \pi_{\text{city}}(\sigma_{\text{position}=\text{“Analyst”}}(R))$$

name	position	city	classification	prov
John	Director	New York	unclassified	x_1
Paul	Janitor	New York	restricted	x_2
Dave	Analyst	Paris	confidential	x_3
Ellen	Field agent	Berlin	secret	x_4
Magdalen	Double agent	Paris	top secret	x_5
Nancy	HR director	Paris	restricted	x_6
Susan	Analyst	Berlin	secret	x_7

city	prov
Paris	$x_3 \otimes x_5$
Berlin	$x_4 \otimes x_7$

What can we do with it?

counting semiring: count the number of times a tuple can be derived, multiset semantics

Boolean semiring: determines if a tuple exists when a subdatabase is selected

security semiring: determines the minimum clearance level required to get a tuple as a result

tropical semiring: minimum-weight way of deriving a tuple (think shortest path in a graph)

Boolean functions: Boolean provenance, as previously defined

integer polynomials: universal provenance, see further

Why-semiring: Why-provenance [Buneman et al., 2001], set of combinations of tuples needed for a tuple to exist

Example of security provenance

$$\pi_{\text{city}}\left[\sigma_{\text{name}<\text{name2}}\left[\pi_{\text{name,city}}(R) \bowtie \rho_{\text{name}\to\text{name2}}(\pi_{\text{name,city}}(R))\right]\right]$$

name	position	city	prov
John	Director	New York	unclassified
Paul	Janitor	New York	restricted
Dave	Analyst	Paris	confidential
Ellen	Field agent	Berlin	secret
Magdalen	Double agent	Paris	top secret
Nancy	HR director	Paris	restricted
Susan	Analyst	Berlin	secret

city	prov
New York	restricted
Paris	confidential
Berlin	secret

Notes [Green et al., 2007]

- Computing provenance has a PTIME data complexity overhead
- Semiring homomorphisms commute with provenance computation: if there is a homomorphism from K to K', then one can compute the provenance in K, apply the homomorphism, and obtain the same result as when computing provenance in K'
- The integer polynomial semiring is universal: there is a unique homomorphism to any other commutative semiring that respects a given valuation of the variables
- This means all computations can be performed in the universal semiring, and homomorphisms applied next
- Two equivalent queries can have two different provenance annotations on the same database, in some semirings

Outline

Semirings with monus [Amer, 1984, Geerts and Poggi, 2010]

- Some semirings can be equipped with a $\ominus$ verifying:
  - $a \oplus (b \ominus a) = b \oplus (a \ominus b)$
  - $(a \ominus b) \ominus c = a \ominus (b + c)$
  - $a \ominus a = \mathbb{0} \ominus a = \mathbb{0}$
- Boolean function semiring with $\wedge\neg$, Why-semiring with $\setminus$, counting semiring with truncated difference...
- Most natural semirings (but not all semirings [Amarilli and Monet, 2016]!) can be extended into semirings with monus
- Sometimes strange things happen [Amsterdamer et al., 2011a]: e.g, $\otimes$ does not always distribute over $\ominus$
- Allows supporting full relational algebra with the $\setminus$ operator, still PTIME
- Semantics for Boolean function semiring coincides with that of Boolean provenance

Difference

Provenance annotations of diff-ed tuples are $\ominus$-ed

Example

$$\pi_{\text{city}}(\sigma_{\textit{ends-with}(\text{position},\text{"agent"})}(R)) \setminus \pi_{\text{city}}(\sigma_{\text{position}=\text{"Analyst"}}(R))$$

name	position	city	classification	prov
John	Director	New York	unclassified	x_1
Paul	Janitor	New York	restricted	x_2
Dave	Analyst	Paris	confidential	x_3
Ellen	Field agent	Berlin	secret	x_4
Magdalen	Double agent	Paris	top secret	x_5
Nancy	HR director	Paris	restricted	x_6
Susan	Analyst	Berlin	secret	x_7

city	prov
Paris	$x_5 \ominus x_3$
Berlin	$x_4 \ominus x_7$

Provenance for aggregates

[Amsterdamer et al., 2011b, Fink et al., 2012]

- Trickier to define provenance for queries with aggregation, even in the Boolean case
- One can construct a K-semimodule $K * M$ for each monoid aggregate M over a provenance database with a semiring in K
- Data values become elements of the semimodule

Example $(\text{count}(\pi_{\text{name}}(\sigma_{\text{city}=\text{``Paris''}}(R)))$

$$x_3 * 1 + x_5 * 1 + x_6 * 1$$

Outline

Application: Probabilistic databases
[Green and Tannen, 2006, Suciu et al., 2011]

- Tuple-independent database: each tuple t in a database is annotated with independent probability $\Pr(t)$ of existing
- Probability of a possible world $D' \subseteq D$:

$$\Pr(D') = \prod_{t \in D'} \Pr(t) \times \prod_{t \in D' \setminus D} (1 - \Pr(t'))$$

- Probability of a tuple for a query q over D:

$$\Pr(t \in q(D)) = \sum_{\substack{D' \subseteq D \\ t \in q(D')}} \Pr(D')$$

- If $\Pr(x_i) := \Pr(x_i)$ where x_i is the provenance annotation of tuple x_i then $\Pr(t \in q(D)) = \Pr(\mathrm{prov}_{q,D}(t))$
- Computing the probability of a query in probabilistic databases thus amounts to computing Boolean provenance, and then computing the probability of a Boolean function
- Also works for more complex probabilistic models

Example of probability computation

name	position	city	classification	prov	prob
John	Director	New York	unclassified	x_1	0.5
Paul	Janitor	New York	restricted	x_2	0.7
Dave	Analyst	Paris	confidential	x_3	0.3
Ellen	Field agent	Berlin	secret	x_4	0.2
Magdalen	Double agent	Paris	top secret	x_5	1.0
Nancy	HR director	Paris	restricted	x_6	0.8
Susan	Analyst	Berlin	secret	x_7	0.2

city	prov
New York	$x_1 \lor x_2$
Paris	$x_3 \lor x_5 \lor x_6$
Berlin	$x_4 \lor x_7$

Example of probability computation

name	position	city	classification	prov	prob
John	Director	New York	unclassified	x_1	0.5
Paul	Janitor	New York	restricted	x_2	0.7
Dave	Analyst	Paris	confidential	x_3	0.3
Ellen	Field agent	Berlin	secret	x_4	0.2
Magdalen	Double agent	Paris	top secret	x_5	1.0
Nancy	HR director	Paris	restricted	x_6	0.8
Susan	Analyst	Berlin	secret	x_7	0.2

city	prov	prob
New York	$x_1 \vee x_2$	$1-(1-0.5)\times(1-0.7)=0.85$
Paris	$x_3 \vee x_5 \vee x_6$	1.00
Berlin	$x_4 \vee x_7$	$1-(1-0.2)\times(1-0.2)=0.36$

Outline

Provenance

Probabilistic Databases

Implementation

- Representation Systems for Provenance
- ProvSQL: Status
- Implementation Challenges

Conclusion

Representation systems

- In the Boolean semiring, the counting semiring, the security semiring: provenance annotations are elementary
- In the Boolean function semiring, the universal semiring, etc., provenance annotations can become quite complex
- Needs for compact representation of provenance annotations
- Lower the provenance computation complexity as much as possible

Provenance formulas

- Quite straightforward
- Formalism used in most of the provenance literature
- PTIME data complexity
- Expanding formulas (e.g., computing the monomials of a $\mathbb{N}[\mathcal{X}]$ provenance annotation) can result in an exponential blowup

Example

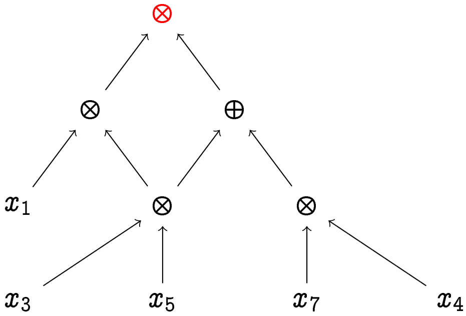
Is there a city with both an analyst and an agent, and if Paris is such a city, is there a director in the agency?

$$((x_3 \otimes x_5) \oplus (x_4 \otimes x_7)) \otimes ((x_3 \otimes x_5) \otimes x_1)$$

Provenance circuits [Deutch et al., 2014, Amarilli et al., 2015]

- Use arithmetic circuits (Boolean circuits for Boolean provenance) to represent provenance
- Every time an operation reuses a previously computed result, link to the previously created circuit gate
- Allow linear-time data complexity of provenance computation when restricted to bounded-treewidth databases [Amarilli et al., 2015] (MSO queries for Boolean provenance, positive relational algebra queries for arbitrary semirings)
- Formulas can be quadratically larger than provenance circuits for MSO formulas, ($\log\log$)-larger for positive relational algebra queries [Wegener, 1987, Amarilli et al., 2016]

Example provenance circuit



OBDD and d-DNNF

- Various subclasses of Boolean circuits commonly used:

  OBDD: Ordered Binary Decision Diagrams

  d-DNNF: deterministic Decomposable Negation Normal Form

- OBDDs can be obtained in PTIME data complexity on bounded-treewidth databases [Amarilli et al., 2016]
- d-DNNFs can be obtained in linear-time data complexity on bounded-treewidth databases
- Application: probabilistic query evaluation in linear-time data complexity on bounded-treewidth databases (d-DNNF evaluation is in linear-time)

Outline

Desiderata for a provenance-aware DBMS

- Extends a widely used database management system
- Easy to deploy
- Easy to use, transparent for the user
- Provenance automatically maintained as the user interacts with the database management system
- Provenance computation benefits from query optimization within the DBMS
- Allow probability computation based on provenance
- Any form of provenance can be computed: Boolean provenance, semiring provenance in any semiring (possibly, with monus), aggregate provenance, where-provenance, on demand

ProvSQL: Provenance within PostgreSQL (1/2)

[Senellart et al., 2018]

- Lightweight extension/plugin for PostgreSQL $\geq$ 9.5 (tested against all versions – upgrade to a new version typically takes a couple of hours)
- Provenance annotations stored as Universally Unique Identifiers (UUIDs), in an extra attribute of each provenance-aware relation
- UUIDs of base tuples randomly generated; UUIDs of query results generated in a deterministic manner
- A provenance circuit relating UUIDs of elementary provenance annotations and arithmetic gates stored in shared memory of the DBMS (or on disk)
- All computations done in the universal semiring (more precisely, with monus, in the free semiring with monus; for where-provenance, in a free term algebra)

ProvSQL: Provenance within PostgreSQL (2/2)

[Senellart et al., 2018]

- Query rewriting (after parsing, before planning) to automatically compute output provenance attributes in terms of the query and input provenance attributes:
  - Duplicate elimination (DISTINCT, set union) results in aggregation of provenance values with $\oplus$
  - Cross products, joins results in combination of provenance values with $\otimes$
  - Difference rewritten in a join, with combination of provenance values with $\ominus$
- Additional circuit gates on projection, join for support of where-provenance
- Probability computation from the provenance circuits, via various methods (naive, sampling, compilation to d-DNNFs, tree decomposition)

ProvSQL: Current status

- Supported SQL language features:
  - Regular SELECT-FROM-WHERE queries (aka conjunctive queries with multiset semantics)
  - JOIN queries (regular joins and outer joins; semijoins and antijoins are not currently supported)
  - SELECT queries with nested SELECT subqueries in the FROM clause
  - GROUP BY queries
  - SELECT DISTINCT queries (i.e., set semantics)
  - UNION's or UNION ALL's of SELECT queries
  - EXCEPT queries
  - Aggregate queries (terminal, for simple aggregates)
- Try it (and see a demo) from `https://github.com/PierreSenellart/provsql`

Other databases with provenance management

- Older probabilistic database systems can compute some forms of provenance (especially, Boolean provenance); but tied to specific version of PostgreSQL (8.3), hard to deploy

  Trio: http://infolab.stanford.edu/trio/ [Benjelloun et al., 2006]

  MayBMS: http://maybms.sourceforge.net/ [Huang et al., 2009]

- Perm https://github.com/IITDBGroup/perm [Glavic and Alonso, 2009] now obsolete system for provenance management; also tied to PostgreSQL 8.3
- ORCHESTRA https://www.cis.upenn.edu/~zives/orchestra/ [Green et al., 2010] Java front end to DBMS with provenance support; not maintained
- GProM http://www.cs.iit.edu/~dbgroup/projects/gprom.html [Arab et al., 2018] similar to ProvSQL (though no probabilistic database capabilities), overlap of features; middleware

Outline

Challenges

- Low-level access to PostgreSQL data structures in extensions
- No simple query rewriting mechanism
- SQL is much less clean than the relational algebra
- Multiset semantics by default in SQL
- SQL is a very rich language, with many different ways of expressing the same thing
- Inherent limitations: e.g., no aggregation within recursive queries
- Implementing provenance computation should not slow down the computation too much – but provenance optimization loses some optimizations
- User-defined functions, updates, arithmetic, etc.: unclear how provenance should work

Provenance computation vs query optimization

- Adding computation of provenance disables some potential optimizations, e.g., hash-based duplicate elimination replaced by a sort-based group-by when a `SELECT DISTINCT` is rewritten into a `GROUP BY` with aggregation
- Computation in a universal algebra (e.g., universal semiring) makes it impossible to benefit of optimizations specific to a given semiring (e.g., absorptivity): generality vs efficiency
- Trade-off between simplifying the provenance circuit and spending as little overhead as possible on provenance computation

How to store provenance?

- Provenance stored as a circuit, but where?
- Two circuit storage modes implemented in ProvSQL:

  As a table of the DBMS: reliable, neat, benefits of the query optimizer, but slow: every queries turns into a series of updates

  In main (shared) memory of the DBMS: much faster, but no logging, resilience to failure, etc.

- Logical thing to do: circuit asynchronously stored on disk, in-memory cache of recent information, but requires a lot of engineering

Efficient probabilistic query evaluation

Safe queries:
- ProvSQL focuses on extensional approach to probabilistic query evaluation: provenance computation, followed by compilation to d-DNNF
- Makes it impossible to apply efficient algorithms for safe queries in probabilistic databaes (safe plans)
- Ideally: explaining intensional approach in the extensional one (see [Monet, 2020] and work in progress by Monet et al.)

Bounded-tw data: maintain a tree decomposition of data to efficiently evaluate queries on them?

Hard query/data: rely on knowledge compilers (see next slide)

Knowledge compilation for probabilistic query evaluation

- Tools such as c2d, d4, dsharp: can be very efficient for compiling (some) Boolean functions to d-DNNF, for weighted model counting of Boolean functions
- These tools expect a formula in CNF, for historical reasons
- Possible to compile an arbitrary circuit to a CNF in linear time (with additional variables) but a lot of the structure is lost in the process
- Typical easy provenance (disjunction of independent events, read-once formula, etc.) become hard instances for knowledge compilers after compilation to CNF

Outline

Provenance

Probabilistic Databases

Implementation

Conclusion

Database Provenance [Senellart, 2017]

- Quite rich foundations of provenance management:
  - Different types of provenance
  - Semiring formalism to unify most provenance forms
  - (Partial) extensions for difference, recursive queries, aggregation, updates [Bourhis et al., 2020]; to other data models
  - Compact provenance representation formalisms
  - Complexity results, classification of queries/databases for which probabilistic query evaluation is tractable [Dalvi and Suciu, 2012, Amarilli et al., 2016]
  - Connections with the field of knowledge compilation [Amarilli et al., 2020]
- ProvSQL: aim at concrete, efficient, usable implementation of all of this!

Many things to do

Usability: Support for larger subset of SQL, utility functions, better interface, documentation, ability to restrict to specific semirings

Efficiency: Benchmarks, optimizations of provenance and probability computation, scalability, manipulate circuit both on disk and in main memory

Knowledge compilation: closer integration with knowledge compilers

More complete probabilistic query evaluation: implementation of safe query plans, continuous probability distributions

Use cases: Work with users, provide semirings that implement useful behavior (e.g., the semiring of unions of real intervals for temporal databases)

Collaborators welcome!

ProvSQL tutorial:

`https://github.com/PierreSenellart/provsql/tree/master/doc/tutorial`

Bibliography I

Antoine Amarilli and Mikaël Monet. Example of a naturally ordered semiring which is not an m-semiring. `http://math.stackexchange.com/questions/1966858`, 2016.

Antoine Amarilli, Pierre Bourhis, and Pierre Senellart. Provenance circuits for trees and treelike instances. In *Proc. ICALP*, pages 56–68, Kyoto, Japan, July 2015.

Antoine Amarilli, Pierre Bourhis, and Pierre Senellart. Tractable lineages on treelike instances: Limits and extensions. In *Proc. PODS*, pages 355–370, San Francisco, USA, June 2016.

Antoine Amarilli, Florent Capelli, Mikaël Monet, and Pierre Senellart. Connecting knowledge compilation classes and width parameters. *Theory Comput. Syst.*, 64(5):861–914, 2020. doi: 10.1007/s00224-019-09930-2. URL `https://doi.org/10.1007/s00224-019-09930-2`.

Bibliography II

K. Amer. Equationally complete classes of commutative monoids with monus. *Algebra Universalis*, 18(1), 1984.

Yael Amsterdamer, Daniel Deutch, and Val Tannen. On the limitations of provenance for queries with difference. In *TaPP*, 2011a.

Yael Amsterdamer, Daniel Deutch, and Val Tannen. Provenance for aggregate queries. In *PODS*, 2011b.

Bahareh Sadat Arab, Su Feng, Boris Glavic, Seokki Lee, Xing Niu, and Qitian Zeng. GProM - A swiss army knife for your provenance needs. *IEEE Data Eng. Bull.*, 41(1):51–62, 2018.

Omar Benjelloun, Anish Das Sarma, Alon Halevy, and Jennifer Widom. ULDBs: Databases with uncertainty and lineage. In *VLDB*, pages 953–964, 2006.

Bibliography III

Pierre Bourhis, Daniel Deutch, and Yuval Moskovitch. Equivalence-invariant algebraic provenance for hyperplane update queries. In *SIGMOD*, pages 415–429. ACM, 2020. doi: 10.1145/3318464.3380578. URL `https://doi.org/10.1145/3318464.3380578`.

Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. Why and where: A characterization of data provenance. In *Database Theory - ICDT 2001, 8th International Conference, London, UK, January 4-6, 2001, Proceedings.*, 2001.

Nilesh Dalvi and Dan Suciu. The dichotomy of probabilistic inference for unions of conjunctive queries. *J. ACM*, 59(6), 2012.

Daniel Deutch, Tova Milo, Sudeepa Roy, and Val Tannen. Circuits for Datalog provenance. In *ICDT*, 2014.

Bibliography IV

Robert Fink, Larisa Han, and Dan Olteanu. Aggregation in probabilistic databases via knowledge compilation. *Proceedings of the VLDB Endowment*, 5(5):490–501, 2012.

Floris Geerts and Antonella Poggi. On database query languages for k-relations. *J. Applied Logic*, 8(2), 2010.

Boris Glavic and Gustavo Alonso. Perm: Processing provenance and data on the same data model through query rewriting. In *ICDE*, pages 174–185, 2009.

Todd J. Green and Val Tannen. Models for incomplete and probabilistic information. *IEEE Data Eng. Bull.*, 29(1), 2006.

Todd J Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *PODS*, 2007.

Bibliography V

Todd J. Green, Grigoris Karvounarakis, Zachary G. Ives, and Val Tannen. Provenance in ORCHESTRA. *IEEE Data Eng. Bull.*, 33(3):9–16, 2010. URL `http://sites.computer.org/debull/A10sept/green.pdf`.

Jiewen Huang, Lyublena Antova, Christoph Koch, and Dan Olteanu. MayBMS: a probabilistic database management system. In *SIGMOD*, pages 1071–1074, 2009.

Tomasz Imieliński and Jr. Lipski, Witold. Incomplete information in relational databases. *J. ACM*, 31(4), 1984.

Mikaël Monet. Solving a special case of the intensional vs extensional conjecture in probabilistic databases. In Dan Suciu, Yufei Tao, and Zhewei Wei, editors, *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2020, Portland, OR, USA, June 14-19, 2020*, pages 149–163. ACM, 2020.

doi: 10.1145/3375395.3387642. URL
`https://doi.org/10.1145/3375395.3387642`.

Pierre Senellart. Provenance and probabilities in relational databases: From theory to practice. *SIGMOD Record*, 46(4), December 2017.

Pierre Senellart, Louis Jachiet, Silviu Maniu, and Yann Ramusat. ProvSQL: provenance and probability management in postgresql. In *VLDB*, 2018. Demonstration.

Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. *Probabilistic Databases*. Morgan & Claypool, 2011.

Ingo Wegener. *The Complexity of Boolean Functions*. Wiley, 1987.