Data sharing among large number of diverse data sources (high variety)
- Sites can have different schemas or even data models, and viewpoints on “truth”

- Sites contribute and import (map) large volumes of data

- Need to handle frequent updates to local and imported data and mappings efficiently (high velocity)

- Big Data Analytics: quality of results only as high as that of input data, need to determine what to trust
“Where Did this Data Come from and how?”

• A common set of questions:
  – Which sources did the data originate from?
  – What operations were used to create and propagate the data?
  – How can we assess trust, data quality etc based on this information?

• Data provenance captures the relationships between items in data instances created through declarative queries or views
  – Different from workflow provenance (e.g., [OPM], [PROV-O]) which captures procedural code and usually treats operations as black boxes due to their complexity

Main topics of this talk

• What data provenance models are there? What data models and query language operators can they capture? How do they compare to each other?

• How can data provenance support assessment of various dimensions of data quality and help in dealing with the 3V’s? What systems and applications take advantage of this?

• What are the benefits of data provenance in Big Data settings, what are the challenges introduced by the 3V’s, and how can we deal with them?
Outline

- Data provenance models for positive relational algebra queries
- Applications of data provenance
- Extensions to the theoretical framework
- Benefits and challenges of data provenance on Big Data

An Example Data Sharing Scenario:
Collaborative Data Sharing in ORCHESTRA [VLDB07]

How do we record provenance for the operations prescribed by these mappings (join, union)?
We’ll adopt two different viewpoints through the talk:

- **Provenance graphs:**
  - Indicates base tuples
  - Multiple incoming edges in a tuple node represent union
  - Multiple incoming edges in mapping nodes represent joins

- **Data provenance as annotations:**
  - The theoretical foundations
  - Standard algebraic identities hold on K-annotated relations iff $(K,\oplus,\otimes,0,1)$ is a commutative semiring
  - Use semiring of polynomials (equivalent to provenance graphs) over base tuple ids as the abstract data provenance model
Semirings unify commonly-used database semantics involving annotations

Standard database models
- set semantics
- bag semantics

Trust, security
- boolean trust, derivability [VLDB07]
- ranked trust [SIGMOD10]
- confidentiality [Foster+08]

Uncertainty, incompleteness
- incomplete DBs [Imielinski+84]
- probabilistic DBs [Fuhr+97]
- ranks, scores [Talukdar+08]

Example: computing ranked (dis)trust annotations through provenance

Provenance polynomials abstract calculations in all commutative semirings

- B does not trust U at all
- B trusts its own data more than Gs

Trust policies
- 0: most trusted, ∞: untrusted
- ⊕: min, ⊗: +

Record data provenance (i.e., abstract annotations) during query evaluation
Evaluate different trust policies or various other annotations through provenance later, often on small subset of query results, without recomputation
- Especially important for Big Data, due to high volume and velocity
Hierarchy of relational abstract provenance models [Green11]

Example: \(2p^2r + pr + 5r^2 + s\)

A path downward from \(K_1\) to \(K_2\) indicates that we can compute \(K_2\) from \(K_1\).

Outline

- Data provenance models for positive relational algebra queries
- Applications of data provenance
  - Provenance querying and annotation computations
  - Uses in research prototypes and commercial systems
- Extensions to the theoretical framework
- Benefits and challenges of data provenance on Big Data
Storing provenance in relations
(Orchestra [VLDB07, SIGMOD10] – LogicBlox [Datalog12])

- Similar to storing graphs in edge relations, but here mapping nodes have multiple “input” and “output” tuples: hyperedges
- Use tuple values (keys) as ids for base tuples

Example of provenance query for ranked (dis)trust computation [SIGMOD10]

Find derivations of B(3,2) from base data

B does not trust U at all
B trusts its own data more than G’s
What is the trust rank of B(3,2)?
ProQL syntax for ranked (dis)trust assessment [SIGMOD10]

EVALUATE RANK OF {
  FOR [B $x] \leftarrow []
  WHERE $x.id = 3$ AND $x.nam = 2$
  INCLUDE PATH [$x] \leftarrow []
  RETURN $x$
}

ASSIGNING EACH leaf_node $y$
{
  CASE $y$ in G : SET 2
  CASE $y$ in U : SET inf
  DEFAULT : SET 0
}

ASSIGNING EACH mapping $p(z)$
{
  CASE $p = m_2$ : SET 2*$z$
  DEFAULT : SET $z$
}

Provenance enables incremental algorithms for handling updates to data and views

- Updates to source data (incremental view maintenance)
  - Past approaches (DRed [Gupta+93]) over-delete and recompute
  - Use data provenance to determine incrementally if derived tuples should be deleted without recomputation

- Updates to derived data (view update)
  - Past approaches ([Dayal+82]) statically check and reject views that may cause side effects on some inputs
  - Use data provenance to determine at runtime if propagating specific deletions to source tuples will actually cause side effects [WebDB07]

- Updates to views (view adaptation)
  - Can be cast as applications of rewriting queries using materialized views, and data provenance can enable more efficient rewritings [Green+11, Green+12]
Provenance for incremental deletion propagation along unidirectional mappings [VLDB07]

Step 1: Use provenance to find derived tuples which should also be deleted
Step 2: Use provenance to also test other affected tuples for derivability, and delete any not derivable
Step 3: Repeat until fixpoint

Program analysis and debugging in LogicBlox [Datalog+12]

Static (BloxAnalysis)
- Represent Datalog programs using relational predicates
- Use Datalog to query and analyze Datalog programs

Examples of BloxAnalysis queries:
- “get all predicates whose names matches foo and all rules in which those predicates appear in the head”
- “find all predicates that are “reachable” from a certain predicate through rules in the program”

Basis for performing more complex reasoning about Datalog programs in Datalog: dead code detection, clone detection …
Analysis and debugging of declarative programs in LogicBlox

- **Dynamic/runtime:**
  - Record data provenance during program evaluation,
  - Use Datalog programs to explore and query resulting provenance graph.

- [Rugaber+13] describe how this functionality could be exposed in an Interactive Development Environment (IDE) for Datalog for program debugging.

Provenance for debugging in other systems

- **GPad [Koehler+12]** Declarative debugging for Datalog
  - Implemented on top of LogicBlox, taking advantage of BloxAnalysis and provenance recording capabilities
  - Uses Datalog and Statelog to represent and query provenance (firing) graphs, e.g., to compute stage of program evaluation at which each fact was derived

- **SPIDER [Chiticariu+06]** uses a form of data provenance (routes) for debugging schema mappings
  - Compute a single derivation for an output tuple, or enumerate all derivations (when there are finitely many)
Outline

• Data provenance models for positive relational algebra queries

• Applications of data provenance

• Extensions to the theoretical framework
  – Provenance models for other query languages/operators
  – Other related theoretical results

• Benefits and challenges of data provenance on Big Data

Relational difference

• M-semirings [Geerts+10] extend semirings with a monus operator to capture relational difference

• Unfortunately there is no suitable abstract structure that can be used as provenance model for m-semirings e.g., to compute various annotations

• [Amsterdamer+11c] identified further difficulties due to the fact that relational difference satisfies two sets of incompatible equivalences in the set and bag semantics
**Provenance for unordered XML** [Foster+08]

K-UUXQuery: Based on Xquery, contains FOR loops and //

*Main result:* Provenance of K-UUXQueries over unordered XML can still be captured by provenance polynomials, and annotations can be computed through it.

**Provenance for RDF**

- **RDF inference rules** [Flouris+09, Udrea+10, Buneman+11, Zimmerman+12]
  - based on similar algebraic structures (idempotent semirings)

- **SPARQL queries**: main challenge from non-monotone OPTIONAL operator (akin to relational left-outer join)
  - Provenance polynomials can still capture provenance of positive SPARQL queries [IntComp11]
  - OPTIONAL can be encoded through relational difference [Damasio+12] (caveat: problems capturing provenance of relational difference)
  - Semirings with an embedded boolean algebra [ICDT13] can be used to construct a suitable data provenance model for SPARQL queries that can be used e.g., to compute various annotations
Other related theoretical results

- Recursion [PODS07]: Results for positive relational algebra can be extended to Datalog for omega-continuous semirings
  - Provenance games [Zinn+13]: Novel unifying framework for dealing with recursion, negation and “why-not” provenance

- Aggregate operators [Amsterdamer+11a]

- Query containment [Green11,Kostylev+12]

- Minimization [Amsterdamer+11b]

- Factorization [Olteanu+12]

Outline

- Data provenance models for positive relational algebra queries

- Applications of data provenance

- Extensions to the theoretical framework

- Benefits and challenges of data provenance on Big Data
Benefits of data provenance for Big Data (1/2)

• Data provenance is crucial for trusted Big Data Analytics due to 3Vs
  – Can be used to assess quality of data imported from large number of diverse sources, possibly using different data models and query languages (relational, XML, RDF) (high variety)
  – combinations of data and workflow provenance [Acar+10, Amsterdamer+12] may help in also dealing with unstructured data
  – Enables quality assessment at any time, even if sources have changed or are unavailable (due to high velocity)
  – Facilitates more efficient provenance querying and annotation computations for small subsets of data, by avoiding recomputation (infeasible, due to high volume and velocity)

Benefits of data provenance for Big Data (2/2)

– Can be used for assessment of various dimensions of data quality, based of different users’ beliefs, again preventing the need for recomputation

• Data provenance enables efficient incremental algorithms for handling updates to data and views in data sharing systems
  – Such updates are frequent, due to high velocity
  – Avoid redundant computations, enable more flexible propagation and more efficient rewritings

• Supports analysis and debugging of declarative programs
  – Especially useful because large numbers of views/mappings are generated automatically by tools
Challenges and research directions

• Preliminary experiments in [VLDB07, SIGMOD10] have shown feasibility for medium-size data sharing settings, investigate performance and scalability in Big Data settings

Some potential ideas/directions

• Optimizations for provenance querying and annotation computations on Big provenance graphs
  – Indexing (some preliminary work in [SIGMOD10])
  – Leverage distributed techniques (e.g., MapReduce)
  – Partial provenance evaluation (e.g., may only care if provenance evaluates to “non-zero” or is over some threshold, not exact value)

Challenges and research directions

• Techniques to optimize provenance storage overhead, and study trade-off with query performance
  – Storage scheme of [SIGMOD10] is a step in this direction
  – Theoretical results on provenance minimization and factorization
  – Explore compression methods or distributed/cloud-based storage

• Tradeoffs between more expressive provenance models and cost of storage/querying
  – Use provenance models that are as informative as necessary
  – Consider partial provenance information e.g., only involving subset of data or query operators, or ignoring some sources
Acknowledgments

• Work in this area done in collaboration with (and some material in a few slides borrowed from):
  – Val Tannen (University of Pennsylvania)
  – Zack Ives (University of Pennsylvania)
  – Todd J. Green (LogicBlox)
  – Floris Geerts (University of Antwerp)
  – Vassilis Christophides (University of Crete & ICS-FORTH)
  – Irini Fundulaki (ICS-FORTH)

Based on work described in:

[ICDT13] F. Geerts, G. Karvounarakis, V. Christophides and I. Fundulaki: Algebraic Structures for Capturing the Provenance of SPARQL Queries, ICDT 2013
References


[Foster+08] J. N. Foster, T. J. Green, V. Tannen: Annotated XML: queries and provenance. PODS 2008
References


References


[OPM] Open Provenance Model. openprovenance.org


References


Using provenance to avoid side effects in bidirectional update exchange [WebDB08]

- Akin to view update

\[ \text{m}_1: \ \star R(x, y), S(x, z) :- T(x, y, z) \]

Diagram:

- \( R(1, 1), (3, 2) \)
- \( S(1, 1, 4), (1, 2, 4), (3, 3, 5) \)
- \( T(1, 1, 1), (1, 1, 2), (3, 3, 3) \)

\[ m_1 \]
Recomputing materialized instances after changes to mappings and data [Green+12]

- Based on Z-relations [Green+11], where updates are represented as annotations
  - Update application can be expressed as regular query: $R' = R \cup R^\delta$

- View maintenance and view adaptation can then be cast as applications of rewriting queries using materialized views
  - Uses provenance to "separate" disjuncts of a union, or "recover" values projected away and enable new (and possibly more efficient) rewritings
  - DBToaster [Koch+10] uses a similar approach for incremental view maintenance and query evaluation

Source relation: $R$

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>b</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>c</td>
<td>b</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>b</td>
<td>c</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>b</td>
<td>a</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

$R^\delta$

<table>
<thead>
<tr>
<th></th>
<th>b</th>
<th>a</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>b</td>
<td>a</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>+2</td>
<td>c</td>
<td>d</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>