Provenance-aware
Versioned
Dataworkspaces

Xing Niu,
Bahareh Sadat Arab,
Boris Glavic

Dieter Gawlick,
Zhen Hua Liu,
Vasudha Krishnaswamy

Oliver Kennedy

Oracle
Introduction

- Data preparation, curation, and analysis
  - Interactive, iterative process with ample uncertainty
  - What datasources to use?
  - How to clean them?
  - How to integrate heterogeneous sources?
  - Requires a lot of backtracking and propagation of changes
    - e.g., find mistake in a previous curation step and correct it
- How to support user in this process?
Our Vision

• Build a model and system implementing the model that supports:
  
  1) Incremental workflow construction with immediate feedback
     • Any change to a curation workflow is immediately reflected in the data

  2) Regret-free exploration through sandboxing
     • Any past choice can easily be undone/changed
     • Derived data is automatically refreshed

  3) Full accountability through provenance tracking
     • All data and transformations are versioned
     • Workflow provenance as well as fine-grained data provenance
Our Vision

• 4) **Automatic conflict detection and resolution**
  - Detect conflicts during automatic refresh of derived data
  - Provide a toolbox of automated resolution techniques for conflicts

• 5) **Merging of transformation pipelines**
  - Update an analysis result if the input data is refreshed

• 6) **Uncertainty as a first-class concept**
  - Expose and propagate uncertainty that naturally arises in data curation
**Virtual Version Graph Model**

- Version control mechanism for **data** and **transformations**
  - Multiple parallel histories can co-exist
  - Explicit tracking of transformations
  - Automatic refresh of derived data
  - A principled and non-invasive way of changing past transformations
  - A lightweight way to represent data and versions
  - Enables data to be materialized on-demand
PVDs

- **Provenance-aware Versioned Dataworkspaces**
  - A sandboxed environment for data curation and analysis backed up by the VVG model
  - Can be implemented on top of existing data processing platforms (e.g., relational DBMS)
Virtual Version Graph Model

- A directed acyclic hyper-graph
- Nodes are relation versions
- Edges are data transformations
- Two types of edges
  - Derivation hyper-edge (Green edge)
  - Version edge (Red edge)
Alex: “I want to build a workflow to determine the success rate of different treatments for lung cancer.”
Running Example

Alex: “Extract data into table T.”

<table>
<thead>
<tr>
<th>Treatment</th>
<th>Disease</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chemotherapy</td>
<td>Lung Cancer</td>
<td>TRUE</td>
</tr>
<tr>
<td>Chemotherapy</td>
<td>Stomach Cancer</td>
<td>TRUE</td>
</tr>
<tr>
<td>Surgery</td>
<td>Lung Cancer</td>
<td>FALSE</td>
</tr>
<tr>
<td>Radiation</td>
<td>Lung Cancer</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

Relation T

For every transformation Q with input I and output O:
- add node O
- add derivation edge labelled Q from I to O
Running Example

Alex: “Calculate the success rate of different treatment methods for Lung Cancer.”

Query Q2

```
SELECT SUM (CASE WHEN Success = TRUE THEN 1 ELSE 0 END) / count(*)
AS SuccessRate,
FROM T
WHERE Disease = “Lung Cancer”
GROUP BY Treatment
```

Queries are transformations that create new relations

Query Q2
Running Example

Alex: “Calculate the success rate of Lung Cancer.”

Result Relation $S$

<table>
<thead>
<tr>
<th>SuccessRate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/3</td>
</tr>
</tbody>
</table>

For transformation $Q_2$ with input $T$ output $S$

- add node $S$
- add derivation edge labelled $Q_2$ from $T$ to $S$
Alex: "*&@#!, I made a mistake when extracting data from the JSON doc. I retrieved the values of attribute Success from the field Finish in the JSON document."

 Json Document J

<table>
<thead>
<tr>
<th>Treatment</th>
<th>Disease</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chemotherapy</td>
<td>Lung Cancer</td>
<td>TRUE</td>
</tr>
<tr>
<td>Chemotherapy</td>
<td>Stomach Cancer</td>
<td>TRUE</td>
</tr>
<tr>
<td>Surgery</td>
<td>Lung Cancer</td>
<td>FALSE</td>
</tr>
<tr>
<td>Radiation</td>
<td>Lung Cancer</td>
<td>FALSE</td>
</tr>
</tbody>
</table>
Alex: "I correct the query Q1 to Q1'."

When modify existing transformation Q to Q' with input I and output O' (the new version of O)
->
- add node O'
- add derivation edge labelled Q' from I to O'
- add version edge (dash line) from O' to O

<table>
<thead>
<tr>
<th>Treatment</th>
<th>Disease</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chemotherapy</td>
<td>Lung Cancer</td>
<td>FALSE</td>
</tr>
<tr>
<td>Chemotherapy</td>
<td>Stomach Cancer</td>
<td>TRUE</td>
</tr>
<tr>
<td>Surgery</td>
<td>Lung Cancer</td>
<td>TRUE</td>
</tr>
<tr>
<td>Radiation</td>
<td>Lung Cancer</td>
<td>TRUE</td>
</tr>
</tbody>
</table>
Running Example

Alex: “I want the derived relations can be automatic refreshed.”

Automatic refresh of derived relations
Create new versions of relations (S) derived from modified relation (T).
- In this case, create new version (S’)

• =>
  - add node S’
  - add derivation edge labelled Q from T’ to S’
  - add version edge (dash line) from S’ to S

Result Relation S’

SuccessRate

2/3
Running Example

• **Conflict**
  - Retrieve values of attribute **Success** ← **Patient**
  - Success: “**TRUE**” or “**FALSE**”
  - **Patient**: Just **strings**
  - Running **Q2** → **Conflicts**

```
{"Treatment" : [
  {
    "Patient": "John",
    "Disease": "Lung Cancer",
    "Doctor": "Xing",
    "Treatment": "Chemotherapy",
    "Suc:" false,
    "Finish:" true
  },
  ...
]}
```

### Strings

- Treatment
  - Chemotherapy
  - Surgery
  - Radiation

- Disease
  - Lung Cancer
  - Stomach Cancer

- Success
  - John
  - Bob
  - Kile
  - Bill

- **true** or **false**
Alex: “I got a conflict when doing automatic refresh.”

Detecting and Dealing with conflicts
Automatic refresh => ill-defined relation versions
- Mark this relation as invalid
- Make semi-automatic and automatic conflict detection
- Provide available resolution strategies

In this case, fix the relation $T'$ firstly, then do automatic refresh (Q2) based on the fixed relation $F$.

- add node $F$ (fixed relation)
- add node $S'$
- add derivation edge labelled Fix from $T$ to $F$
- add derivation edge labelled Q2 from $F$ to $S'$
- add version edge (dash line) from $S'$ to $S$
Related Work

Version Control Systems
- git
- mercurial
- FOSSIL
- DATA HUB
- TOASTER

Scientific workflow management systems

Workflow and Database Provenance
- Taverna
- myGrid
- VT
- VisTrails
- Kepler
PVD

- An interface similar to iPython
- System maintains a VVG for the users actions
- Relation versions can be stored in, e.g., a DBMS
- Visualizations are represented as special VVG nodes
PVD building blocks

- **Data curation with lenses** [1]
  - Powerful uncertainty aware data curation operations
  - Uses probabilistic database techniques to keep track of uncertainty

- **Provenance tracking and reenactment for updates** [2]
  - Declarative replay technique
  - Retroactively compute provenance for updates
  - Translates an update into an equivalent query

Implementation Challenges

• Strategies for materializing
  ✤ Which relation versions, when and how

• Methods for compressing VVGs

• Incremental view maintenance techniques

• Conflicts and merging VVGs
Conclusion

• A novel version model (**VVG**) and system vision (**PVDs**)
  
  • Keep all track of users’ operations and data provenance
  
  • Supports exploratory application of data curation and analysis operators
  
• **Features**
  
  • Simple and clean model
  • Automatic refresh
  • Past transformations can be modified
  • Automatic conflict detection (and resolution)
Questions?

• My Webpage
  • http://www.cs.iit.edu/~dbgroup/people/xniu.php

• Our Group’s Webpage
  • http://cs.iit.edu/~dbgroup/research/index.html

• GProM
  • http://www.cs.iit.edu/~dbgroup/research/gprom.php

• Mimir
  • http://odin.cse.buffalo.edu/research/mimir/

• Vizier (The ODIn Lab)
  • http://www.vizierdb.info