Repairing Inconsistent Data Warehouses

Mónica Caniupán
Universidad del Bio-Bio
mcaniupa@ubiobio.cl

Joint work with: Loreto Bravo (Universidad de Concepción, Chile)
Carlos Hurtado (Universidad Adolfo Ibáñez, Chile)
Alejandro Vaisman (Universidad de la República, Uruguay)

Outline
1. Motivation and preliminaries
2. Repairing dimension instances
3. Complexity
4. Computing repairs
5. R-repairs
6. Conclusions

Multidimensional Data Warehouses (MDW)
- A historical data repository
- Integrates data from multiple sources
- Used for analysis and decision support
- Very different from an operational DB:
  - Non-volatile
  - Summarized
  - Data identified with a particular time period

Operational DBs vs Data warehouses

<table>
<thead>
<tr>
<th>Standard DB</th>
<th>Data Warehouse</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mostly updates</td>
<td>Mostly reads</td>
</tr>
<tr>
<td>Many small transactions</td>
<td>Queries are long and complex</td>
</tr>
<tr>
<td>Mb - Gb of data</td>
<td>Gb - Tb of data</td>
</tr>
<tr>
<td>Current snapshot</td>
<td>Historical data</td>
</tr>
<tr>
<td>Raw data</td>
<td>Summarized, reconciled data</td>
</tr>
<tr>
<td>Thousands of users</td>
<td>Hundreds of users (e.g., decision-makers, analysts)</td>
</tr>
</tbody>
</table>
DWH Architecture

Information Sources

External Sources

ETL: Extract, Transform and Load

Operational DBs

Data Warehouse

OLAP

Data Mining

Data Visualization

Reporting

DWH Architecture

Multidimensional Data Model

- Attributes of different nature
  - Facts: numerical data, e.g. quantities, units, ...
  - Dimensions: data coordinates
- Facts can be seen as points in a multidimensional space determined by the dimensions

Attributes of different nature

Facts: numerical data, e.g. quantities, units, ...
Dimensions: data coordinates
Facts can be seen as points in a multidimensional space determined by the dimensions

An online article repository

Fact table:

<table>
<thead>
<tr>
<th>Article</th>
<th>Date</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>01-01-2007</td>
<td>3</td>
</tr>
<tr>
<td>A2</td>
<td>01-01-2007</td>
<td>2</td>
</tr>
<tr>
<td>A3</td>
<td>02-01-2007</td>
<td>5</td>
</tr>
</tbody>
</table>

Dimension schemas:

Article dimension instance:

More formally

Dimension schema $S = (\mathcal{C}, \prec)$

- $\mathcal{C}$: is a set of categories
- $\prec$: child/parent relation between categories

The transitive closure of $\prec$ is denoted by $\prec^*$
More formally

- Dimension instance $D=(M, <)$ over schema $(C,<^*)$
  - $M$: contains the elements of each category
  - $<$: roll-up relation between elements

- The transitive closure of $<$ is denoted by $<^*$

Homogenous Dimensions

- Homogenous instance: every element has an ancestor element in its ancestor categories defined in the schema

Answering queries

- Total number of downloads grouped by Area:
  - $\text{SELECT R.Area, SUM(D.N)} \text{ FROM Downloads D, R.Area WHERE D.Article = R.Article GROUP BY R.Area}$

- Answers: $<\text{CS,8}> <\text{Bio,7}>$

Using pre-computed answers

- What if we have already computed the aggregation of downloads for Subject or ISSN?
- Efficiency of query answering could be improved!!
Using pre-computed answers

- If we use DownloadsSubject the optimized query would be:

  Q: SELECT R.Area, SUM(D.N) 
  FROM DownloadsSubject D, πArea ⋈ Subject R 
  WHERE D.Subject = R.Subject 
  GROUP BY R.Area 

<table>
<thead>
<tr>
<th>DownloadsSubject</th>
<th>R.Area</th>
<th>Subject</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subject</td>
<td>DB</td>
<td>CS</td>
<td>Gen</td>
</tr>
<tr>
<td>N</td>
<td>8</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

- Answers: <CS,8> <Bio,7>

- The same answer would be obtained from DownloadsISSN

Answering queries from a non-strict instance

- Total number of downloads grouped by Area:

  Q: SELECT R.Area, SUM(D.N) 
  FROM Downloads D, πArea ⋈ Article R 
  WHERE D.Area = R.Area 
  GROUP BY R.Area 

<table>
<thead>
<tr>
<th>Downloads</th>
<th>Article</th>
<th>Date</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS</td>
<td>A1</td>
<td>01-01-2007</td>
<td>3</td>
</tr>
<tr>
<td>Bio</td>
<td>A2</td>
<td>01-01-2007</td>
<td>2</td>
</tr>
<tr>
<td>Bio</td>
<td>A3</td>
<td>01-02-2007</td>
<td>5</td>
</tr>
<tr>
<td>Bio</td>
<td>A4</td>
<td>02-01-2007</td>
<td>5</td>
</tr>
</tbody>
</table>

- Answers: <CS,15> <Bio,7>

Using pre-computed answers

- Aggregation of downloads for Subject or ISSN:

<table>
<thead>
<tr>
<th>DownloadsSubject</th>
<th>DownloadsISSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subject</td>
<td>ISSN</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>DB</td>
<td>0392</td>
</tr>
<tr>
<td>Gen</td>
<td>1234</td>
</tr>
</tbody>
</table>

- From them we would get:
  - <CS,8><Bio,7> from DownloadsSubject
  - <CS,15> from DownloadsISSN

- Both are incorrect answers!!

Summarizability

- A dimension instance is summarizable if pre-computed answers can be used to correctly compute answers

- This property is of great importance for MDWs
  - Efficiency!

- A dimension is summarizable if it is both:
  - homogenous
  - strict

[Lenz and Shoshani, 1997]
Strictness
- each element can have no more than one ancestor in each category
- Strictness ensures that the roll-up relations between categories are functional

Strictness
- Its crucial for efficiency of MDWs
  - It makes the use of pre-computed answers possible
- Most commercial MDW systems assume that dimensions are strict
  - Pre-computed answers are used without checking for strictness
- Strictness should be enforced by developers when:
  - data is loaded
  - every time the dimensions are updated
  - Updates: result of changes in data sources or modification to the business rules

Non-strict dimensions
- If at design time, or after an update a dimension becomes non-strict:
  - It needs to be repaired!
- Not feasible to repair by hand
- We need to develop tools to aid with the process of repairing non-strict dimensions

Repairing dimension instances
- A Repair $D'$ of a non-strict dimension $D$ should:
  - Have the same schema as $D$
  - Be homogenous and strict
  - Contain the same elements in each category as $D$
    - If elements were added:
      - what's their meaning?
    - If elements were removed:
      - data of fact tables could be lost
- Repairs are obtained by insertion and deletions of edges (roll-up relations)
Some possible repairs for $D$

Are some better than others?

Preferred repairs?
- We want repairs that are as close as possible to the original instance
- Minimal repair:
  - repair that minimizes the number of insertion and deletions of edges

Preferred repairs?
- We want repairs that are as close as possible to the original instance
- Minimal repair:
  - repair that minimizes the number of insertion and deletions of edges

Example
- Sometimes only deletions are required:

  Non-strict Instance
  Repair $D'$
  Repair $D''$
Repairs

- Number of minimal repairs?

  \[ \text{All} \]
  \[ \text{B} \]
  \[ \text{A} \]

  - It is possible to have an exponential number!

Complexity

- It is relevant then to study the problem of finding one

  **Theorem 1**
  **Problem:** Is there a repair of a dimension \( D \) at a distance smaller than \( k \)?
  **Complexity:** NP-complete

- The proof uses a reduction from the set covering problem which is NP-complete

---

Complexity

- A direct consequence is:

  **Theorem 2**
  **Problem:** Is \( D' \) a minimal repair of \( D \)?
  **Complexity:** co-NP-complete

- Even if we had a repair it is not tractable to check if it is minimal!

Complexity

- Are there any good news?

  \[ \text{A repair of linear schemas can be found in polynomial time on the size of the dimension} \]
Computing repairs

- As shown, it can be expensive!
- We need an expressive language:
  - Datalog programs with negation under stable models semantics
- Several implementations:
  - DLV \([\text{Leone et al. 2006, TOCL}]\)
  - Smodels \([\text{Syrjnen et al. 2001, LNCS 2173}]\)
- We will define a repair program in this language to represent and compute minimal repairs of a dimension

Predicates used in the repair program

<table>
<thead>
<tr>
<th>Atom</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(C(a))</td>
<td>element (a) belongs to category (C) in dimension instance (D)</td>
</tr>
<tr>
<td>(R(a, b, C_1, C_2))</td>
<td>(a) rolls-up to (b) in dimension instance (D), (a \in C_1) and (b \in C_2)</td>
</tr>
<tr>
<td>(R'(a, b, C_1, C_2))</td>
<td>(a) rolls-up to (b) in the repair, (a \in C_1) and (b \in C_2)</td>
</tr>
<tr>
<td>(RT'(a, b, C_1, C_2))</td>
<td>transitive closure of (R')</td>
</tr>
<tr>
<td>(Ins(a, b, C_1, C_2))</td>
<td>the edge ((a, b)) with (a \in C_1) and (b \in C_2) was added to the repair</td>
</tr>
<tr>
<td>(Del(a, b, C_1, C_2))</td>
<td>the edge ((a, b)) with (a \in C_1) and (b \in C_2) was removed from the repair</td>
</tr>
</tbody>
</table>

Repair program facts

<table>
<thead>
<tr>
<th>R</th>
<th>Area</th>
<th>Journal</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_1</td>
<td>TODS Article Journal</td>
<td></td>
</tr>
<tr>
<td>A_2</td>
<td>DKE Article Journal</td>
<td></td>
</tr>
<tr>
<td>DB</td>
<td>Journal Subject</td>
<td></td>
</tr>
<tr>
<td>DKE</td>
<td>Gen Journal ISSN</td>
<td></td>
</tr>
<tr>
<td>TODS 0362</td>
<td>Journal ISSN</td>
<td></td>
</tr>
<tr>
<td>DKE 1234</td>
<td>Journal ISSN</td>
<td></td>
</tr>
<tr>
<td>DB</td>
<td>CS Subject Area</td>
<td></td>
</tr>
<tr>
<td>Gen</td>
<td>Bio Subject Area</td>
<td></td>
</tr>
<tr>
<td>0362</td>
<td>CS ISSN Area</td>
<td></td>
</tr>
<tr>
<td>1234</td>
<td>CS ISSN Area</td>
<td></td>
</tr>
<tr>
<td>CS</td>
<td>all Area All</td>
<td></td>
</tr>
<tr>
<td>Bio</td>
<td>all Area All</td>
<td></td>
</tr>
</tbody>
</table>

Repair program rules

\[
R'(x, y', z, \text{Journal}) \iff R(x, y, z, \text{Journal}, \text{Journal}(y')) \land \text{choice}(x, \text{Journal}(y')), x \neq y. \\
R'(x, y', z, \text{Subject}) \iff R(x, y, z, \text{Subject}, \text{Subject}(y')), \text{choice}(x, \text{Subject}(y')), x \neq y. \\
R'(x, y', z, \text{ISSN}) \iff R(x, y, z, \text{ISSN}, \text{ISSN}(y')), \text{choice}(x, \text{ISSN}(y')), x \neq y. \\
R'(x, y', z, \text{All}) \iff R(x, y, z, \text{All}, \text{All}(y')), \text{choice}(x, \text{All}(y')), x \neq y. \\
R'(x, y, n_1, n_2) \iff R'(x, y, m_1, n_2). \\
RT'(x, z, n_1, n_3) \iff RT'(x, y, n_1, n_2), R'(y, z, n_2, n_3). \\
\Leftrightarrow RT'(x, y, n_1, n_2), RT'(x, z, n_1, n_2), y \neq z. \\
Ins(x, y, n_1, n_2) \iff R'(x, y, n_1, n_2), \text{not } R(x, y, n_1, n_2). \\
Del(x, y, n_1, n_2) \iff R(x, y, n_1, n_2), \text{not } R'(x, y, n_1, n_2). \\
\Leftrightarrow Ins(x, y, n_1, n_2)[1:1]. \\
\Leftrightarrow Del(x, y, n_1, n_2)[1:1].
\]
Stable models

\[ S_{M_1} = \{ \text{Del(DKE, Gen, Subject, Area)}, \text{Ins(DKE, DB, Subject, Area)} \} \]
\[ \text{Cost([weight : level])} \leq [2 : 1] \]

\[ S_{M_2} = \{ \text{Del(Gen, Bio, Subject, Area)}, \text{Ins(Gen, CS, Subject, Area)} \} \]
\[ \text{Cost([weight : level])} \leq [2 : 1] \]

\[ S_{M_3} = \{ \text{Del(1234, CS, ISSN, Area)}, \text{Ins(1234, Bio, ISSN, Area)} \} \]
\[ \text{Cost([weight : level])} \leq [2 : 1] \]

R-repairs

- A dimension becomes inconsistent after a reclassification of edges

\[ D' \text{ is an } r \text{-repair of } D: \]
  - \( D' \) is a repair
  - \( D' \) contains the changes in a set \( R \) of reclassifications
  - \( D' \) is minimal if it is minimal among all the repairs for \( D \)

- Only \( D_1 \) and \( D_3 \) are \( r \)-repairs of \( D \) since they keep the update
Results for R-repairs

- A good result:
  - The problem of deciding if there exists an r-repair \( D' \) of \( D_u \) with respect to \( \Sigma \), such that \( \text{dist}(D_u, D') \leq k \) is NP-complete.
  - The problem of deciding if \( D' \) is an r-repair of \( D_u \) is co-NP-complete.

Polynomial Algorithms

- It is possible to find efficient algorithms to find r-repairs for many common real-world DW dimensions.
- We study r-repairs for multiple path dimensions, restricted to the ones containing at most one conflicting level that becomes inconsistent with respect to strictness after one reclassify operation.

Heuristics

- We present heuristics and algorithms that find an r-repair for a dimension instance under reclassification in polynomial time.
- They are based on two concepts:
  - Ensuring that during the repair computation we do not generate new conflicts, that is, we do not generate new conflicting paths.
  - Taking the best decision at each step.

No-new conflicts

- Elements involve in inconsistencies \( a_2, a_5 \).
Take the best decision

- Changing the parent of \( b_i \) is not an option, since it is too expensive (many edges have to be modified).

Conclusions

- Both heuristics need to be implemented
- Experiments must be performed
- We need help!!!