Data Integration: Schema Mapping

Jan Chomicki

University at Buffalo

Data integration

Data sources

- data in any format/data model

Wrappers

- typically: relational or XML
- data/query translation, data publishing
- using source query interfaces

Mediators

- restructuring, merging, reconciliation,...
- eager or lazy

Relational data integration

Data integration system

- target (integrated) schema, incl. integrity constraints
- one or more source schemas, incl. constraints
- assertions (or queries) relating the contents of the target to the contents of the source(s)

Data integration

- source schema given
- target schema and/or assertions (queries) to be constructed
- target instance corresponding to the given source instance may or may not be materialized

Data exchange

- source and target schemas given
- assertions to be constructed
- target instance needs to be materialized

Problems

Schema matching

Establishing correspondences between elements of the source and target schemas.

Schema mapping

Generation of assertions (queries) from schema correspondences.

Data reconciliation

- underspecification: selecting the target instance (uniqueness, nulls)
- overspecification: what if target constraints cannot be satisfied?
- ambiguity: object identification (record linkage)

Schematic discrepancies

- correspondences mix schema/instance elements
- beyond SQL queries/first-order assertions

Finding a "best" match

- start with some initial match and try to improve it
- rank the results

Similarity Flooding

- matching schemas represented as labelled directed graphs
- relational, XML, ontologies,...

Pairwise connectivity graph $PCG(A, B)$

- A and B are graphs to be matched
- $N(PCG(A, B)) = \{(x, y) \mid x \in N(A), y \in N(B)\}$
- $E(PCG(A, B)) = \{((x, y), p, (x', y')) \mid (x, p, x') \in E(A), (y, p, y') \in E(B)\}$

Similarity Flooding algorithm

Induced propagation graph $IPG(A, B)$

- $N(IPG(A, B)) = \{(x, y) \mid x \in N(A), y \in N(B)\}$
- $E(IPG(A, B)) = \{((x, y), (x', y')) \mid \exists p\ ((x, y), p, (x', y')) \in E(PCG(A, B)) \lor ((x', y'), p, (x, y)) \in E(PCG(A, B))\}$
- propagation coefficients $w((x, y), (x', y'))$ calculated and used to label edges

Algorithm

1. construct an initial mapping (similarity measure) σ^0, consisting of weighted pairs of nodes in graphs A and B
2. construct the mapping σ^i based on neighborhood information
3. repeat Step 2 for $i + 1$ if necessary
4. filter the result

Adjustment step

$$\begin{aligned}\sigma^{i+1}(x,y) = \quad & \sigma^{i}(x,y) + \sum_{(a_u,p,x)\in E(A),(b_u,p,y)\in E(B)} \sigma^{i}(a_u,b_u)\cdot w((a_u,b_u),(x,y)) \\ + \quad & \sum_{(x,p,a_v)\in E(A),(y,p,b_v)\in E(B)} \sigma^{i}(a_v,b_v)\cdot w((a_v,b_v),(x,y)).\end{aligned}$$

The new values are normalized to $[0,1]$ after each iteration.

Termination

- when the changes to the mapping are below a threshold
- after a fixed number of iterations
- guaranteed for strongly connected graphs.

Schema mapping in CLIO

Setting

- source database S, target database T
- input:
  - schema correspondences
  - filters on attributes
  - source and target constraints: keys, foreign keys
- output:
  - schema mapping: query or assertions relating S and T
  - query: union of conjunctive queries

Schema mapping

1. creating candidate sets of correspondences: each target attribute is mentioned at most once
2. finding join paths in each candidate set
3. ranking join paths
4. covering all the correspondences

Finding join paths

- using foreign keys
- query history
- discovering joinable columns

Ranking join paths

1. prefer paths through foreign keys
2. if multiple such paths, prefer one with a filter
3. least number of dangling tuples

Computing covers

- cover:
  - set of candidate sets
  - every correspondence belongs to some candidate set
  - minimal
- ranking covers:
  - smaller number of candidate sets
  - more target attributes

Creating mapping query

For each candidate set V of the selected cover

SELECT *attributes in V*
FROM *source relations in the join paths for V*
WHERE *filters and join conditions from the join paths*

For the entire selected cover

Compute the UNION of the SELECT blocks.