Generic Schema Matching with Cupid

J. Madhavan, P. Bernstein, E. Rahm

presentation by Viorica Botea

Overview
- Schema matching problem
- Cupid approach
- Experimental results
- Conclusion

The Schema Matching Problem
- Given two input schemas in any data model and, optionally, auxiliary information, compute a mapping between schema elements of the two input schemas

Classification Criteria for Matching Techniques
- Schema vs. Instance based
- Element vs. Structure granularity
- Linguistic based
- Constraint based
- Matching cardinality
- Auxiliary information
- Individual vs. Combinatorial

The Cupid Approach
- Includes automated linguistic-based matching
- Is both element-based and structure-based
- Is biased toward similarity of atomic elements
- Exploits keys, referential constraints and views

The Cupid Approach (2)
- Is schema-based
- Models the interconnected elements of a schema as a schema tree

PO
- PoShipTo
  - City
  - Street
- POBillTo
  - City
  - Street
- POLines
  - Item
  - Qty
  - UOM
The Cupid Approach (3)

- Consists of two phases:
  1. Linguistic matching: based on names, data types, domains, etc.; the result is a linguistic similarity coefficient, \( l_{sim} \)
  2. Structure matching: based on the similarity of element contexts or vicinities; the result is a structural similarity coefficient, \( s_{sim} \)

**Weighted similarity:**
\[
 w_{sim} = w_{struct} * s_{sim} + (1-w_{struct}) * l_{sim}
\]
where \( w_{struct} \) is a constant in the range \([0,1]\).

Linguistic Matching

1. **Normalization:**
   - Tokenization: parsing names into tokens based on punctuation, case, etc.
   - Expansion: identifying abbreviations and acronyms
   - Elimination: discharging prepositions, articles, etc.

2. **Categorization:** elements are clustered into categories, based on their data types, schema hierarchies and linguistic content

3. **Comparison:** linguistic similarity coefficients are computed between schema elements by comparing the tokens extracted from their names

Result: a table of \( l_{sim} \) coefficients in the range \([0,1]\), with 1 indicating a perfect match

Structure Matching

- The *TreeMatch* algorithm:
  a) Atomic elements (leaves) are similar if they are individually (linguistic and data type) similar and their respective vicinities (ancestors and siblings) are similar
  b) Two non-leaf elements are similar if they are linguistically similar and the subtrees rooted at the two elements are similar
  c) Two non-leaf elements are structurally similar if their leaves sets are highly similar, even if their immediate children are not

**Matching Schema Trees**

```
TreeMatch((SourceTree S, TargetTree T))
for each s in S where s.t are leaves
set sum(s,t) = datatype_comparability(s,t)
S’ = post-order(S), T’ = post-order(T)
for each s in S’
for each t in T’
   compute ssim(s, t) = structural-similarity(s, t)
   wsim(s, t) = wstruct * ssim(s, t) + (1 - wstruct) * lsim(s, t)
   if wsim(s, t) > th_accept
      increase-struct-similarity(leaves(s), leaves(t), c_increase)
   if wsim(s, t) < th_reject
      decrease-struct-similarity(leaves(s), leaves(t), c_decrease)
```

Matching Schema Trees (2)

- The structural similarity of two leaves is initialized to the type compatibility of their data types
- The structural similarity of two non-leaf elements is:
\[
 s_{sim}(s, t) = \frac{|\{x \in \text{leaves}(s) \land \exists y \in \text{leaves}(t), \text{stronglink}(x, y)\}|}{|\text{leaves}(s) \cup \text{leaves}(t)|}
\]
A leaf in a schema has a strong link to a leaf in other schema if their weighted similarity exceeds a threshold \( th_{accept} \).

Mappings

- After the matching schema process is completed, a set of similarity coefficients is returned
- For leaf elements, if \( w_{sim}(s, t) \geq th_{accept} \), a mapping between \( s \) and \( t \) is returned
- For non-leaf elements, a second post-order traversal of the two schemas is done, to re-compute similarities; after this, a mapping is generated as in the case of leaves
Extending to General Schemas

- To capture more semantics as shared types and referential constraints, the schema is extended to a rooted graph structure, whose nodes are schema elements.

**Schema Graphs**

- Types of relationships among interconnected schema elements:
  1. **Containment**: each element is contained by exactly one other element (e.g. a table contains its columns)
  2. **Aggregation**: allow multiple parents for an element (e.g. a compound key aggregates columns of a table)
  3. **IsDerivedFrom**: abstracts IsA and IsTypeOf relationships to model shared types

Matching Shared Types

- When shared types are present, there can be several paths from the root to an element $e$.
- Each path defines a context, and thus is a candidate for a different mapping.
- To materialize all the paths, the schema is converted to a tree.

**Matching Shared Types (2)**

```
schema_tree = construct_schema_tree(schema.root, NULL)
construct_schema_tree(Schema Element current_se, Schema Tree Node current_stn)
if current_se is the root or current_se was reached through a containment relationship
  if current_se is not instantiated then return current_stn
  new_stn = new schema tree node corresponding to current_se
  set new_stn as a child of current_stn
  current_stn = new_stn
for each outgoing containment or isDerivedFrom relation
  new_se = schema element that is the target of the relationship
  construct_schema_tree(new_se, current_stn)
return current_stn
```

Example

Referential Constraints

- Are represented by `RefInt` elements, that aggregates the source and the reference.
Matching Referential Constraints (2)

Goals:
- Improve the structural matching
- Enable discovery of mappings between join view in a schema and a single table or other join views in the second schema

Example

Augmenting the schema tree

Other Cupid’s Features

Optionality: nodes are divided in optional and required; optional leaves are not considered when the structural similarity of a node is computed

Views: are treated as referential constraints

Lazy expansion: compares elements at the schema graph before converting it to a tree

Pruning leaves: consider only nodes in a subtree of depth $k$, in order to reduce the computation time

Experimental Conclusions

- Tokenization step and using a thesaurus play a crucial role in linguistic matching
- Using leaves in the schema tree allows to match similar schemas that have different nesting
- Context-dependent mappings are useful when inferring different mappings for the same element in different contexts
- The use of linguistic mapping without structure mapping can lead to false mappings

Future Work

- Using schema annotations (textual descriptions of schema elements in the data dictionary) for linguistic matching
- Automatic tuning of the control parameters
Conclusion

- Schema matching problem
- Cupid approach
- Experimental results