

Leveraging Query Logs for Schema Mapping Generation in U-MAP

Taras Kinash

David R. Cheriton School of Computer Science
University of Waterloo

Problem Overview

Schema Mapping problem is characterized by a set:

$$(S, T, \Sigma_S, \Sigma_T, V, \Sigma_{ST})$$

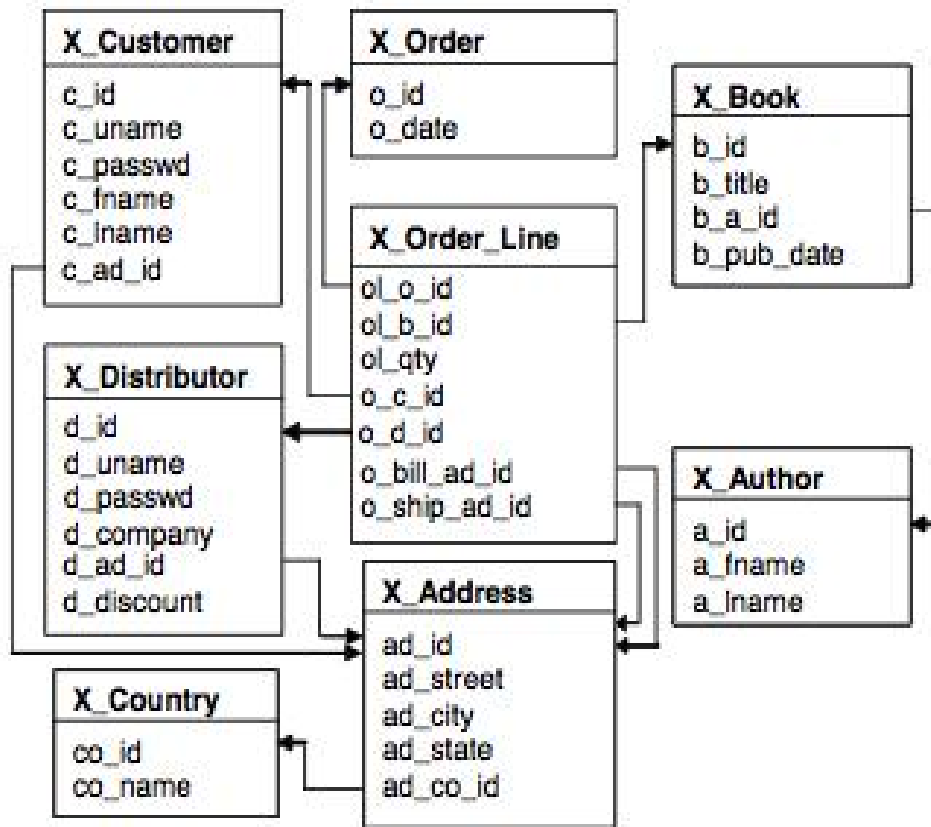
INPUT:

- S/T – source/target schema
- Σ_S / Σ_T – constraints defined on S/T respectively
(i.e. foreign keys)
- V – attribute correspondences between S and T

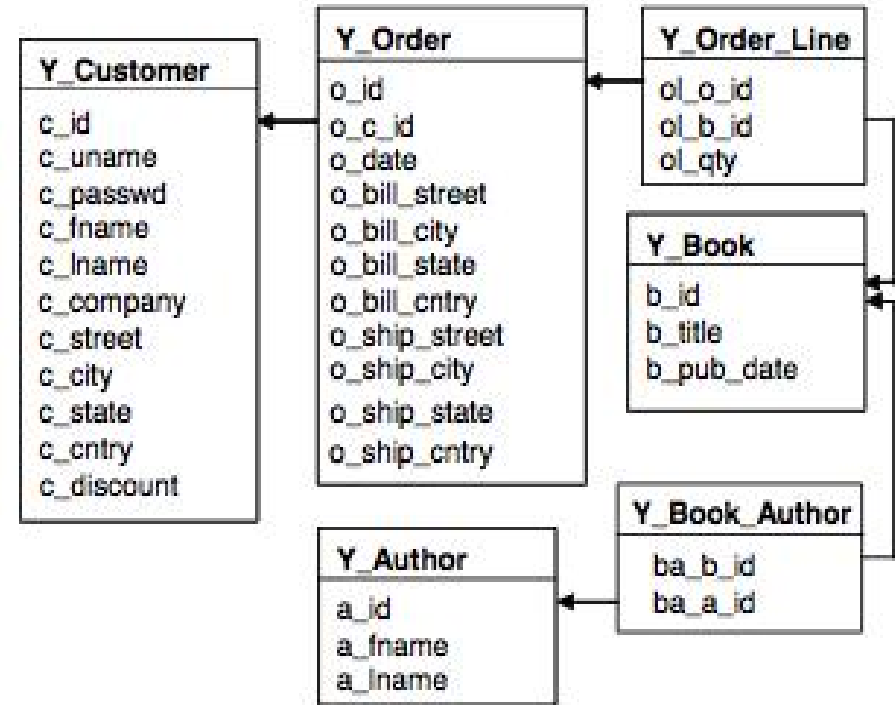
OUTPUT:

- Σ_{ST} – a set of tuple generating dependencies,
representing the schema mapping.

Example



(a) X-Schema



(b) Y-Schema

for a in X_Author , b in X_Book
where $(a.a_id = b.b_a_id)$
exists a' in Y_Author , b' in Y_Book , ba' in Y_Book_Author
where $(a'.a_id = ba'.ba_a_id \wedge b'.b_id = ba'.ba_b_id)$
with $(a'.a_fname = a.a_fname \wedge a'.a_lname = a.a_lname$
 $\wedge b'.b_title = b.b_title \wedge b'.b_pub_date = b.b_pub_date)$

Existing Solutions

Referential Integrity Constraints (RIC) approach:

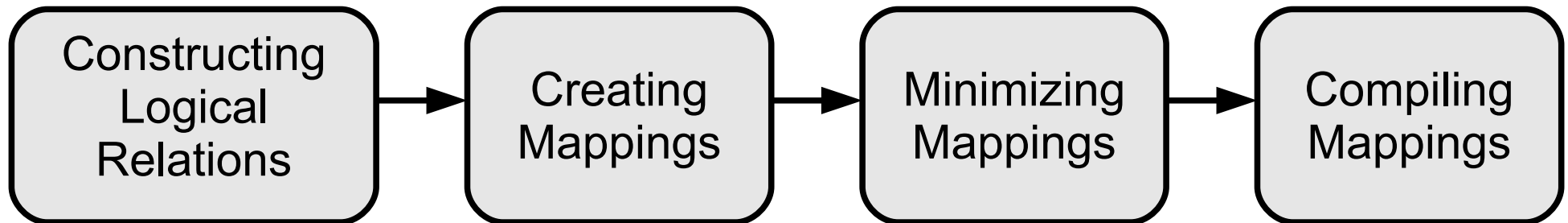
- Exploring referential constraints of the schema to compute join paths.
- Clio (IBM & UofT), MQG...

Semantic approach:

- Utilize available semantic information from conceptual schemas (i.e. ER-diagrams or UML).
- Reduced to sub-graph matching.

RIC-approach

Proceeds in four steps:



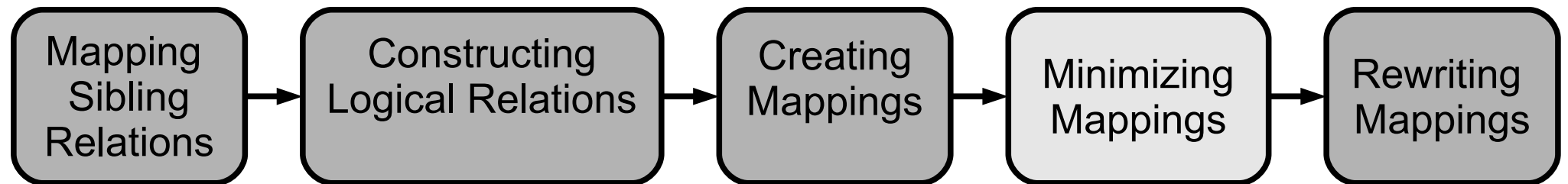
- *Logical Relations* – for each table, join it with all tables it references; continue iteratively.
- *Creating Mappings* – chose pairs of logical relations from S and T , which cover part of V – these would be mappings.
- *Minimizing Mappings* – discard mappings which are subsumed or implied by other mappings.
- *Compiling Mappings* – convert mappings into executable script.

Issues with Existing Solutions

- Handling Sibling Relations
- Incomplete Logical Relations
- Handling Attribute Correspondence Conflicts

Overview of the U-MAP

U-MAP builds on the RIC-approach, in an attempt to resolve these issues.



Handling Sibling Relations

- Merging Sibling Relations
 - Identify pairs of potential sibling relations.
 - Filter out pairs that are not siblings or non-overlapping siblings (Query Logs are utilized here).
 - For each applicable pair, combine the two relations into a “super-relation”.
 - Update the schema and query logs to reflect this “super-relation”
- Rewriting Mappings
 - Replace “super-relations” appearing in resulting mappings by the original sibling relations.

Incomplete coverage of Logical Relations

RIC-approach only considers forward foreign key expansion when building logical relations.

How to determine when reverse expansion is meaningful?

Utilize information from query logs!

Incomplete coverage of Logical Relations cont'd

Aggressive Chase Algorithm:

- Scan query log, and fill out Forward-Reverse (FR) index.
- For each relation R of the schema, create logical relation in an alternating manner:
 - (Forward) Join all the tables that R references, to create intermediate logical relation LR'
 - (Reverse) For each join condition used in previous step, look up the reverse join condition in FR-index, and join LR' with other tables using these new conditions.
 - Continue in this alternating manner, until no further forward expansion is possible

Incomplete coverage of Logical Relations cont'd

Example:

SCHEMA (partial):

RELATIONS: $A(\underline{a}_1, a_2), B(\underline{b}_1, b_2), C(\underline{c}_1, c_2)$

FOREIGN KEYS:

$A(a_2)$ references $B(b_1)$

$C(c_2)$ references $B(b_1)$

QUERY:

```
SELECT *  
FROM A, B, C  
WHERE A.a2 = B.b1 AND B.b1 = C.c2
```

Incomplete coverage of Logical Relations cont'd

FR-index:

Forward	Reverse
(A.a ₂ , B.b ₁)	(C.c ₂ , B.b ₁)
(C.c ₂ , B.b ₁)	(A.a ₂ , B.b ₁)

Building Logical Relations starting from relation *A*:

Phase 1 – forward: $LR' = A \text{ join } B$

Phase 2 – reverse: $LR = LR' \text{ join } C$

Resolving Correspondence Conflicts

Formulation:

For two logical relations with sets of attributes $\{a_1, \dots, a_n\}$ and $\{b_1, \dots, b_m\}$ respectively. The set of correspondences contains tuples of the form (a_i, b_j) . Pair of correspondences are conflicting if they are of the form:

$$\langle (a_i, b_{j1}), (a_i, b_{j2}) \rangle \text{ or } \langle (a_{i1}, b_j), (a_{i2}, b_j) \rangle$$

b_{j1} & b_{j2} and a_{i1} & a_{i2} , respectively, are conflicting attributes.

Observation:

Conflicting attributes result from using same concepts but in different contexts.

Resolving Correspondence Conflicts - Solution

Solution in two phases:

Attribute Grouping:

- group attributes by context.

Usage-based Conflict Resolution:

- Utilize existing techniques for usage-based schema matching to come up with the mappings.

Definitions

Logical Attribute Path (*LAP*):

A_k – tables in the schema for k in $[0, K]$; A_0 – initial table

a_k – primary key of table A_k ; a'_k – foreign key in table A_k

$LAP(a) = (A_0.a'_{i0}=A_1.a_{i1})....(A_k.a_{ik} = A_{k+1}.a'_{k+1})...A_K.a$

Query Attribute Path (*QAP*):

Defined in the same way as LAP, but for a query instead of a *LR*.

Minimal Distinguishing LAP (*MDLAP*):

$MDLAP(a)$ is a minimal suffix of $LAP(a)$, such that there does not exist attribute b in *LR*, where $MDLAP(a)$ is a suffix of $LAP(b)$.

Attribute Grouping

$LAP(a)$ – context of attribute a .

Context of attributes a and b are compared in terms of common prefix of $LAP(a)$ and $LAP(b)$.

Goal of generating attribute groups:

- (1) All attributes in the same groups are non-conflicting with each other
- (2) For any two attributes a and b in groups G_1 and G_2 respectively, common prefix of $LAP(a)$ and $LAP(b)$ cannot be longer than common prefix of $LAP(a)$ and $LAP(c)$, for attribute c in group G_1 .
- (3) The number of groups that satisfy (1) and (2) must be minimal.

Attribute Grouping Algorithm

Algorithm proceeds in the following way:

- For each conflicting attribute a in a logical relation, insert $LAP(a)$ into a prefix tree. Each internal node represents a set of attributes
- Perform depth first search on a prefix tree.
- At each node, check if the corresponding set of attributes are conflicting.
 - If yes – continue with search
 - If no – consider this set to be a group.

Usage-based Conflict Resolution

Key point: consider only mappings where no two attributes in same group on one side are mapped to two attributes in different groups on the other side.

Want to use existing techniques for schema matching based on usage information (i.e. examining query logs)

BUT, must be context-aware when examining query logs!

Usage-based Conflict Resolution cont'd

For each attribute a in each query of the log, compute $QAP(a_q)$; this may not be unique.

For each attribute a_l in logical relation, compute $MDLAP(a_l)$.

If $MDLAP(a_l)$ is a suffix of $QAP(a_q)$ – a_q is considered to be an occurrence of a_l .

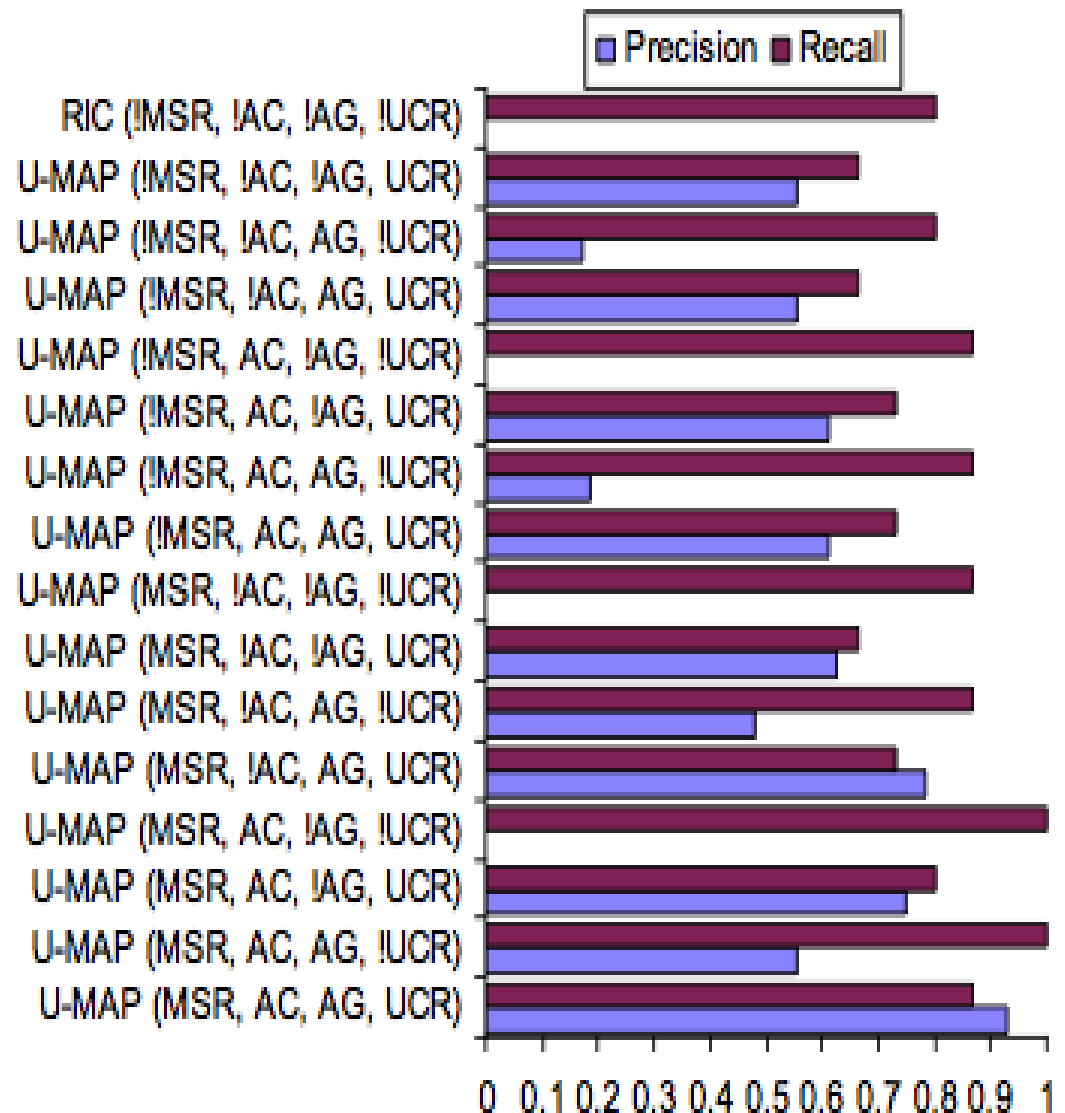
Now, can apply existing usage-based attribute matching techniques.

U-MAP Evaluation - Setup

- System: modular implementation of each feature
(*MSR*, *AC*, *AG*, *UCR*)
- Dataset: bookstore schema (modified *TPC-W*)
- Query Logs: modified *TPC-W* queries.
- Measures: precision, recall.

U-MAP Evaluation - Experiments

- Best – when all features are “on”
- Disabling *AG* and *UCR* drastically reduces precision.
- Generally – each feature adds some benefit to system.



U-MAP Evaluation – Life Science Scenario

Consider a large scale database (35 tables, 273 attributes, 73 query templates)

Issues addressed by U-MAP are likely to occur in reality:

- Aggressive Chase – 106 possible opposing/reverse references, 3 are interesting
- Conflict Resolution – 1267 conflicting attributes.

Conclusion

- U-MAP system is based on a classical RIC-approach.
- U-MAP addresses three shortcomings of RIC-approach:
 - Overlapping sibling relations
 - Meaningful “reverse” references
 - Conflicting attribute correspondences
- Precision and recall for fully implemented U-MAP is at least as good as for RIC-approach.

Critique & Discussion

- More than two overlapping sibling relations?
- Experiments:
 - Only one dataset used for experiments
 - “Artificial” dataset
 - Comparison with other techniques (i.e. semantic)?
- Query Logs:
 - May be unavailable
 - Optimal size?
 - Pre-process (i.e. annotate in some manner)?