Schema Matching

Partly based on slides by AnHai Doan
Motivation: Data Integration

New faculty member

Find houses with 2 bedrooms priced under 200K

realestate.com
homeseekers.com
homes.com
Architecture of Data Integration System

Find houses with 2 bedrooms priced under 200K

mediated schema

source schema 1
realestate.com

source schema 2
homeseekers.com

source schema 3
homes.com
Semantic Matches between Schemas

Mediated-schema

<table>
<thead>
<tr>
<th>price</th>
<th>agent-name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1 match</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1-1 match

complex match

homes.com

<table>
<thead>
<tr>
<th>listed-price</th>
<th>contact-name</th>
<th>city</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>320K</td>
<td>Jane Brown</td>
<td>Seattle</td>
<td>WA</td>
</tr>
<tr>
<td>240K</td>
<td>Mike Smith</td>
<td>Miami</td>
<td>FL</td>
</tr>
<tr>
<td>......</td>
<td>......</td>
<td>......</td>
<td>......</td>
</tr>
</tbody>
</table>

4
Schema Matching is Ubiquitous

- Fundamental problem in numerous applications
- Databases
  - data integration
  - data translation
  - schema/view integration
  - data warehousing
  - semantic query processing
  - model management
  - peer data management
- AI
  - knowledge bases, ontology merging, information gathering agents, ...
- Web
  - e-commerce
  - marking up data using ontologies (e.g., on Semantic Web)
Schema Matching is Difficult

- Schema & data never fully capture semantics!
  - not adequately documented
  - schema creator has retired to Florida!
- Must rely on clues in schema & data
  - using names, structures, types, data values, etc.
- Such clues can be unreliable
  - same names => different entities: area => location or square-feet
  - different names => same entity: area & address => location
- Intended semantics can be subjective
  - house-style = house-description?
  - military applications require committees to decide!
- Cannot be *fully* automated, needs user feedback!
Source Modeling vs. Schema Matching

- **Schema Matching/Mapping**
  - Align schemas between data sources
  - Assumes static sources and complete access to data

- **Source modeling**
  - Incrementally build models from partial data (e.g., web services, html forms, programs)
  - Model not just the fields but the source types and even the function of a source
  - Support richer source models (a la Semantic Web)
Overview

- Survey of schema matching
  - Review of existing methods
    - Matchers use information in the schema, data instances, or both
    - Use manually specified rules or learn rules from the data
    - Users evaluate the best matches to generate mappings

- **iMap: Discovering Complex Semantic Matches between Database Schemas**
  - Semi-automatically discovers 1:1 and complex matches
  - Combines multiple searchers
  - Includes domain knowledge to facilitate search
Schema Mapping

- *Schema* is a set of elements connected by some structure
- *Mapping*: certain elements of schema S1 are mapped to certain elements in S2.
- Mapping expression specifies how S1 and S2 elements are related
  - Simple
    - Home.price = Property.listed-price
  - Complex
    - Concatenate(Home.city, Home.state) = Property.address

<table>
<thead>
<tr>
<th>S1 elements</th>
<th>S2 elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Home</td>
<td>Property</td>
</tr>
<tr>
<td>price</td>
<td>listed-price</td>
</tr>
<tr>
<td>agent-name</td>
<td>contact-name</td>
</tr>
<tr>
<td>city</td>
<td>address</td>
</tr>
<tr>
<td>state</td>
<td></td>
</tr>
</tbody>
</table>
Current State of Affairs

- Finding semantic mappings is now a key bottleneck!
  - largely done by hand
  - labor intensive & error prone
  - data integration at GTE [Li & Clifton, 2000]
    - 40 databases, 27,000 elements, estimated time: 12 years

- Will only be exacerbated
  - data sharing becomes pervasive
  - translation of legacy data

- Need semi-automatic approaches to scale up!

- Many research projects in the past few years
  - Databases: IBM Almaden, Microsoft Research, BYU, George Mason, U of Leipzig, U Wisconsin, NCSU, UIUC, Washington, ...
  - AI: Stanford, Karlsruhe University, NEC Japan, ...
Variety of Schema Matching Approaches

- Match algorithm can consider
  - *Instance* data – i.e., data contents
  - *Schema* information or metadata

- Match can be performed on
  - *Individual elements* – e.g., attributes
  - *Schema structure* – combination of elements

- Match algorithm can use
  - *Language-based* approaches – e.g., based on names or textual descriptions
  - *Constraint-based* approach – based on keys and relationships

- Match may relate 1 or n elements of one schema to 1 or n elements of another schema
Classification of Schema Matching Approaches

Schema Matching Approaches

Individual matcher approaches

- Schema-only based
  - Element-level
    - Linguistic
    - Constraint-based
  - Structure-level
    - Constraint-based

Instance/contents-based

- Element-level
  - Linguistic
  - Constraint-based

Combining matchers

- Hybrid matchers
- Composite matchers

- Manual composition
- Automatic composition

Further criteria:
- Match cardinality
- Auxiliary information used ...

Sample approaches

- Name similarity
- Description similarity
- Global namespaces
- Type similarity
- Key properties
- Graph matching
- IR techniques (word frequencies, key terms)
- Value pattern and ranges
Match Granularity

- Element- vs structure level
- Element-level matching
  - For each element of S1, determine matching elements of S2
    - \texttt{Home.price=Property.listed-price}
- Structure-level matching
  - Match combinations of elements that appear together
    - \texttt{Home=Property}
- Match takes into account name, description, data type of schema element

<table>
<thead>
<tr>
<th>S1 elements</th>
<th>S2 elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Home</td>
<td>Property</td>
</tr>
<tr>
<td>price</td>
<td>listed-price</td>
</tr>
<tr>
<td>agent-name</td>
<td>contact-name</td>
</tr>
<tr>
<td>city</td>
<td>address</td>
</tr>
<tr>
<td>state</td>
<td></td>
</tr>
</tbody>
</table>
## Match Cardinality

<table>
<thead>
<tr>
<th>Match cardinalities</th>
<th>S1</th>
<th>S2</th>
<th>Match expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:1</td>
<td>Price</td>
<td>Amount</td>
<td>Amount=Price</td>
</tr>
<tr>
<td>n:1</td>
<td>Price, Tax</td>
<td>Cost</td>
<td>Cost=Price*(1+Tax/100)</td>
</tr>
<tr>
<td>1:n</td>
<td>Name</td>
<td>FirstName, LastName</td>
<td>FirstName, LastName=Extract(Name, …)</td>
</tr>
<tr>
<td>n:m</td>
<td>B.Title, B.PuNo, P.PuNo, P.Name</td>
<td>A.Book, A.Publisher</td>
<td>A.Book,A.Publisher=Select B.Title,P.Name, From B,P where B.PuNo=P.PuNo</td>
</tr>
</tbody>
</table>
Linguistic Approaches

- Language-based approaches analyze text to find semantically similar schema elements
  - Schema name matching
    - Equality of names, before and after stemming
    - Equality of synonyms
      - Car=automobile, make=brand
    - Similarity based on edit distance, soundex (how they sound)
      - ShipTo=Ship2, representedBy=representative
  - Description matching
    - Schema contain comments in natural language to explain the semantics of elements
  - Instance-level matching
    - Data content can give insight into the meaning of schema elements
Constraint-based Approaches

- For schema-level matching
  - Schemas often contain constraints to define data types and value ranges, foreign keys, … which can be exploited in matching two schemas

- For instance-level matching
  - Value ranges and averages on numeric elements
  - Character patterns on string fields
Combining Matchers

- Hybrid matcher combines several matching approaches
  - Determine match candidates using multiple criteria or information sources

- Composite matcher combines results of several independently executed matchers
  - Machine learning to combine instance-level matchers or instance and schema-level matchers
LSD: Learning Source Descriptions

- Developed at Univ of Washington 2000-2001
  - AnHai Doan, Pedro Domingos and Alon Halevy
- LSD uses machine learning to match new data source against a global manually-created schema
- Desirable characteristics
  - learn from previous matching activities
  - exploit multiple types of information in schema and data
  - handle user feedback
  - achieves high matching accuracy (66 -- 97%) on real-world data
LSD Approach

1. User
   - manually creates matches for a few sources
   - shows LSD these matches

2. LSD learns from the matches

3. LSD predicts matches for remaining sources
   - Matching approach
     - Composite match with automatic combination of match results
       - Schema-level matchers
         - Names, schema tags in XMLs
       - Instance-level matchers
         - Trained during the preprocessing step to discover characteristic instance patterns and matching rules
         - Learned patterns and rules are applied to match other sources to the global schema
Discussion

- Schema matching techniques line up the elements of one schema with another, or a global schema.
- Matchers use information in the schema, data instances, or both.
  - Use manually specified rules or learn rules from the data.
- **LSD**
  - Learns from previous matching activities.
  - Exploits multiple types of information.
    - By employing multi-strategy learning.
  - Incorporates domain constraints & user feedback.
  - Focuses on 1:1 matches.
- **Next challenge: discover more complex matches!**
  - **iMAP** (Illinois Mapping) system [SIGMOD-04]
  - With Robin Dhamanka, Yoonkyong Lee, Alon Halevy, Pedro Domingos.
iMap: Discovering Complex Semantic Matches between Database Schemas
The iMAP Approach

- For each mediated-schema element
  - searches space of all matches
  - finds a small set of likely match candidates

- To search efficiently
  - employs a specialized searcher for each element type
  - Text Searcher, Numeric Searcher, Category Searcher, ...
The iMAP Architecture [SIGMOD-04]

- **Mediated schema**
- **Source schema + data**

**Searcher**

- \( \text{Searcher}_1 \)
- \( \text{Searcher}_2 \)
- \( \ldots \)
- \( \text{Searcher}_k \)

**Match candidates**

- **Base-Learner**
  - \( \text{Base-Learner}_1 \)
  - \( \ldots \)
  - \( \text{Base-Learner}_k \)

**Meta-Learner**

**Similarity Matrix**

**Match selector**

**Explanation module**

**1-1 and complex matches**

**User**

**Domain knowledge and data**
Candidate Match Generator

- Given target (mediated) schema, generator discovers a small set of candidate matches
- Search through space of possible match candidates
  - Uses specialized searchers
    - Text searchers: know about concat operation
    - Numeric searchers: know about arithmetic operations
  - Each searcher explores a small portion of search space based on background knowledge of operators and attribute types
- System is extensible with additional searchers
  - E.g., Later add searcher that knows how to operate on Address
Searcher

- Search strategy
  - Beam search to handle large search space
  - Uses a scoring function to evaluate match candidate
  - At each level of search tree, keep only $k$ highest-scoring match candidates

- Match evaluation
  - Score of match candidates approximates semantic distance between it and target attribute
  - E.g., concat(city, state) and agent-address
  - Uses machine-learning, statistics, heuristics

- Termination condition – when to stop?
  - Diminishing return
    - Highest scores of beam search do not grow as quickly
**An Example: Text Searcher**

- Find match candidates for **address**
- **Search** in space of all concatenation matches over all string attributes

<table>
<thead>
<tr>
<th>listed-price</th>
<th>agent-id</th>
<th>full-baths</th>
<th>half-baths</th>
<th>city</th>
<th>zipcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>320K</td>
<td>532a</td>
<td>2</td>
<td>1</td>
<td>Seattle</td>
<td>98105</td>
</tr>
<tr>
<td>240K</td>
<td>115c</td>
<td>1</td>
<td>1</td>
<td>Miami</td>
<td>23591</td>
</tr>
</tbody>
</table>

- **Best match candidates for address**
  - (agent-id,0.7), (concat(agent-id,city),0.75), (concat(city,zipcode),0.9)
## iMap Searchers

<table>
<thead>
<tr>
<th>Searcher</th>
<th>Space of candidates</th>
<th>Examples</th>
<th>Evaluation technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>Text attributes of source schema</td>
<td>name=concat(first-name,last-name)</td>
<td>Naïve Bayes and beam search</td>
</tr>
<tr>
<td>Numeric</td>
<td>User supplied matches of past complex matches</td>
<td>list-price=price*(1+tax-rate)</td>
<td>Binning, KL divergence</td>
</tr>
<tr>
<td>Category</td>
<td>Attributes w/less than <em>t</em> distinct values</td>
<td>product-categories=product-types</td>
<td>KL divergence</td>
</tr>
<tr>
<td>Schema mismatch</td>
<td>Source attribute containing target schema info</td>
<td>fireplace=1 if house-desc has “fireplace”</td>
<td>KL divergence</td>
</tr>
<tr>
<td>Unit conversion</td>
<td>Physical quantity attributes</td>
<td>weigh-kg=2.2*lbs</td>
<td>Properties of distributions</td>
</tr>
<tr>
<td>Dates</td>
<td>Columns recognized as ontology nodes</td>
<td>birth-date=b-day/b-month / b-year</td>
<td>Mapping into ontology</td>
</tr>
</tbody>
</table>
Similarity Estimator

- Scores assigned to each candidate match by the Searcher may not be accurate, since it is based on only one type of information

- Measure similarity between candidate match and attribute $t$
  - Uses multiple evaluator modules to suggest scores based on different types of information
  - Combines suggested scores

- Example: name-based evaluator
  - Computes a score of each match candidate based on similarity of its name (including table name) to the name of the target attribute
Match Selector

- However, match with highest similarity score may violate domain integrity constraints
  - Maps two source attributes to target attribute list-price
- Match selector searchers for the best **global match assignment** that satisfies domain constraints
Exploiting Domain Knowledge

- Domain knowledge can help reduce search space, direct search, and prune unlikely matches early
- Types of domain knowledge
  - Domain constraints
    - *name* and *beds* are unrelated → never generate match candidates that combine these attributes
  - Past complex matches in related domains
    - Reuse past matches: e.g., \( \text{price} = \text{pr}^*(1+0.06) \) to produce a template \( \text{VARIABLE}^*(1+\text{CONSTANT}) \) to guide search
  - Overlap data between databases
    - Source and target databases share some data
    - Re-evaluate matches based on overlap data
  - External data supplied by domain experts
    - Can be used to describe the properties of attributes
Empirical Evaluation

- Current iMAP system
  - 12 searchers
- Four real-world domains
  - real estate, product inventory, cricket, financial wizard
  - target schema: 19 -- 42 elements, source schema: 32 -- 44
- Accuracy: 43 -- 92%
- Sample discovered matches
  - agent-name = concat(first-name, last-name)
  - area = building-area / 43560
  - discount-cost = (unit-price * quantity) * (1 - discount)

- More detail in [Dhamanka et. al. SIGMOD-04]
Observations

- Finding complex matches much harder than 1-1 matches!
  - require gluing together many components
  - e.g., \( \text{num-rooms} = \text{bath-rooms} + \text{bed-rooms} + \text{dining-rooms} + \text{living-rooms} \)
  - if missing one component \( \Rightarrow \) incorrect match

- However, even partial matches are already very useful!
  - so are top-k matches \( \Rightarrow \) need methods to handle partial/top-k matches

- Huge/infinite search spaces
  - domain knowledge plays a crucial role!

- Matches are fairly complex, hard to know if they are correct
  - must be able to explain matches

- Human must be fairly active in the loop
  - need strong user interaction facilities

- Break matching architecture into multiple "atomic" boxes!
Finding Matches is only Half of the Job!

- To translate data/queries, need **mappings**, not **matches**

### Schema $S$

**HOUSES**

<table>
<thead>
<tr>
<th>location</th>
<th>price ($)</th>
<th>agent-id</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atlanta, GA</td>
<td>360,000</td>
<td>32</td>
</tr>
<tr>
<td>Raleigh, NC</td>
<td>430,000</td>
<td>15</td>
</tr>
</tbody>
</table>

**AGENTS**

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>city</th>
<th>state</th>
<th>fee-rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>Mike Brown</td>
<td>Athens</td>
<td>GA</td>
<td>0.03</td>
</tr>
<tr>
<td>15</td>
<td>Jean Laup</td>
<td>Raleigh</td>
<td>NC</td>
<td>0.04</td>
</tr>
</tbody>
</table>

### Schema $T$

**LISTINGS**

<table>
<thead>
<tr>
<th>area</th>
<th>list-price</th>
<th>agent-address</th>
<th>agent-name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Denver, CO</td>
<td>550,000</td>
<td>Boulder, CO</td>
<td>Laura Smith</td>
</tr>
<tr>
<td>Atlanta, GA</td>
<td>370,800</td>
<td>Athens, GA</td>
<td>Mike Brown</td>
</tr>
</tbody>
</table>

- **Mappings**
  - `area` = `SELECT location FROM HOUSES`
  - `agent-address` = `SELECT concat(city, state) FROM AGENTS`
  - `list-price` = `price * (1 + fee-rate)`
    - `FROM HOUSES, AGENTS`
    - `WHERE agent-id = id`
Developed at Univ of Toronto & IBM Almaden, 2000-2003
   – by Renee Miller, Laura Haas, Mauricio Hernandez, Lucian Popa, Howard Ho, Ling Yan, Ron Fagin

Given a match
   – \textit{list-price} = \textit{price} \times (1 + \textit{fee-rate})

Refine it into a mapping
   – \textit{list-price} = \text{SELECT} \textit{price} \times (1 + \textit{fee-rate})
     \text{FROM} \text{HOUSES (FULL OUTER JOIN) AGENTS}
     \text{WHERE} \textit{agent-id} = \textit{id}

Need to discover
   – the correct join path among tables, e.g., \textit{agent-id} = \textit{id}
   – the correct join, e.g., full outer join? inner join?

Use heuristics to decide
   – when in doubt, ask users
   – employ sophisticated user interaction methods [VLDB-00, SIGMOD-01]
Clio: Illustrating Examples

Schema S

<table>
<thead>
<tr>
<th>location</th>
<th>price ($)</th>
<th>agent-id</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atlanta, GA</td>
<td>360,000</td>
<td>32</td>
</tr>
<tr>
<td>Raleigh, NC</td>
<td>430,000</td>
<td>15</td>
</tr>
</tbody>
</table>

Schema T

<table>
<thead>
<tr>
<th>area</th>
<th>list-price</th>
<th>agent-address</th>
<th>agent-name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Denver, CO</td>
<td>550,000</td>
<td>Boulder, CO</td>
<td>Laura Smith</td>
</tr>
<tr>
<td>Atlanta, GA</td>
<td>370,800</td>
<td>Athens, GA</td>
<td>Mike Brown</td>
</tr>
</tbody>
</table>

AGENTS

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>city</th>
<th>state</th>
<th>fee-rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>Mike Brown</td>
<td>Athens</td>
<td>GA</td>
<td>0.03</td>
</tr>
<tr>
<td>15</td>
<td>Jean Laup</td>
<td>Raleigh</td>
<td>NC</td>
<td>0.04</td>
</tr>
</tbody>
</table>

Mappings

- area = SELECT location FROM HOUSES
- agent-address = SELECT concat(city, state) FROM AGENTS
- list-price = price * (1 + fee-rate)
  FROM HOUSES, AGENTS
  WHERE agent-id = id
Summary

- Schema matching:
  key to numerous data management problems
  - Much attention in the database, AI, Semantic Web communities
  - Related to ontology matching problem

- Simple problem definition, yet very difficult to do
  - no satisfactory solution yet

- We now understand the problems much better
  - still at the beginning of the journey
  - will need techniques from multiple fields