Source Modeling vs. Schema Matching

• **Schema Matching**
  - 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

- **Learning Semantic Descriptions of Web Information Sources**
  - Integrated approach to learning semantic definitions of a source
  - Needs to invoke the source
  - Learn Datalog description of the source

- **Automatically Constructing Semantic Web Services from Online Sources**
  - Deimos: End to end system for discovering and modeling sources
  - Uses system above to automatically discover semantic definitions of discovered sources
Mediators Require Source Definitions

- New service => no source definition!
- Can we discover a definition automatically?

Query

```sql
SELECT MIN(price)
FROM flight
WHERE depart="LAX"
AND arrive="MXP"
```

Reformulated Query

```java
lowestFare("LAX","MXP")
calcPrice("LAX","MXP","economy")
```

Source Definitions:
- Orbitz Flight Search
- United Airlines
- Qantas Specials

Orbitz Flight Search
United Airlines
Qantas Specials
Alitalia
Inducing Source Definitions by Example

• Step 1: classify input & output semantic types

source1($zip, lat, long) :-
    centroid(zip, lat, long).

source2($lat1, $long1, $lat2, $long2, dist) :-
    greatCircleDist(lat1, long1, lat2, long2, dist).

source3($dist1, dist2) :-
    convertKm2Mi(dist1, dist2).

- Known Source 1
- Known Source 2
- Known Source 3
- New Source 4

source4( $startZip, $endZip, separation)
• Step 1: classify input & output semantic types
• Step 2: generate plausible definitions

source1($zip, lat, long) :-
    centroid(zip, lat, long).
source2($lat1, $long1, $lat2, $long2, dist) :-
    greatCircleDist(lat1, long1, lat2, long2, dist).
source3($dist1, dist2) :-
    convertKm2Mi(dist1, dist2).

source4($zip1, $zip2, dist) :-
    source1(zip1, lat1, long1),
    source1(zip2, lat2, long2),
    source2(lat1, long1, lat2, long2, dist2),
    source3(dist2, dist).

source4($zip1, $zip2, dist) :-
    centroid(zip1, lat1, long1),
    centroid(zip2, lat2, long2),
    greatCircleDist(lat1, long1, lat2, long2, dist2),
    convertKm2Mi(dist1, dist2).
- Step 1: classify input & output semantic types
- Step 2: generate plausible definitions
- Step 3: invoke service & compare output

<table>
<thead>
<tr>
<th>$zip1</th>
<th>$zip2</th>
<th>dist (actual)</th>
<th>dist (predicted)</th>
</tr>
</thead>
<tbody>
<tr>
<td>80210</td>
<td>90266</td>
<td>842.37</td>
<td>843.65</td>
</tr>
<tr>
<td>60601</td>
<td>15201</td>
<td>410.31</td>
<td>410.83</td>
</tr>
<tr>
<td>10005</td>
<td>35555</td>
<td>899.50</td>
<td>899.21</td>
</tr>
</tbody>
</table>

source4($zip1, $zip2, dist):-
  source1(zip1, lat1, long1),
  source1(zip2, lat2, long2),
  source2(lat1, long1, lat2, long2, dist2),
  source3(dist2, dist).

source4($zip1, $zip2, dist):-
  centroid(zip1, lat1, long1),
  centroid(zip2, lat2, long2),
  greatCircleDist(lat1, long1, lat2, long2, dist2),
  convertKm2Mi(dist1, dist2).
Searching for Definitions

- Search space of *conjunctive queries*:
  \[ \text{target}(X) :- \text{source1}(X_1), \text{source2}(X_2), \ldots \]
- For scalability don’t allow negation or union
- Perform Top-Down Best-First Search

1. First sample the New Source
   - Invoke `target` with set of random inputs;
   - Add empty clause to `queue`;
   - While (queue not empty)
     - `v` := best definition from `queue`;
     - For all `(v' in Expand(v))`
       - If (Eval(v') > Eval(v))
         - Insert `v'` into `queue`;

2. Then perform best-first search through space of candidate definitions

*Expressive Language* Sufficient for modeling most online sources
Invoking the Target

Invoke source with *representative* values

- Try randomly generating input tuples:
  - Combine examples of each type
  - Use distribution if available

```
source5( $zip1, $dist1, zip2, dist2)
```

### Randomly Combined Example Values

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;zip1, dist1&gt;</td>
<td>&lt;zip2, dist2&gt;</td>
</tr>
<tr>
<td>&lt;07307, 50.94&gt;</td>
<td>{&lt;07097, 0.26&gt;, &lt;07030, 0.83&gt;, &lt;07310, 1.09&gt;, ...}</td>
</tr>
<tr>
<td>&lt;60632, 10874.2&gt;</td>
<td>{}</td>
</tr>
</tbody>
</table>

Non-empty Result

Empty Result
Invoke source with *representative* values

- Try randomly generating input tuples:
  - Combine examples of each type
  - Use distribution if available

- If *only empty invocations* result
  - Try *invoking other sources* to generate input

- Continue until sufficient non-empty invocations result
Top-down Generation of Candidates

Start with empty clause & generate specialisations by
- Adding one predicate at a time from set of sources
- Checking that each definition is:
  - Not logically redundant
  - Executable (binding constraints satisfied)

```
source5(_,_,_,_).

source5(zip1,_,_,_) :- source4(zip1,zip1,_).
source5(zip1,_,zip2,dist2) :- source4(zip2,zip1,dist2).
source5(_,dist1,_,dist2) :- <(dist2,dist1).
...
```

New Source 5

source5( $zip1,$dist1,zip2,dist2)

University of Southern California
Best-first Enumeration of Candidates

- Evaluate each clause produced
- Then expand best one found so far
- Expand high-arity predicates incrementally

```
source5(zip1,_,zip2,dist2) :- source4(zip2,zip1,dist2).

source5(zip1,dist1,zip2,dist2) :- source4(zip2,zip1,dist2), source4(zip1,zip2,dist1).

source5(zip1,dist1,zip2,dist2) :- source4(zip2,zip1,dist2), <(dist2,dist1).

... 
```
Limiting the Search

• Extremely Large Search space
• Constrained by use of Semantic Types
• Limit search by:
  • Maximum Clause length
  • Maximum Predicate Repetition
  • Maximum Number of Existential Variables
  • Definition must be Executable
  • Maximum Variable Repetition within Literal
• Compare output of clause with that of target.
• Average the results across different input tuples.
Evaluating Candidates II

Candidates may return multiple tuples per input
  • Need measure that compares sets of tuples!

<table>
<thead>
<tr>
<th>Input</th>
<th>Target Output</th>
<th>Clause Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;$zip1, $dist1&gt;</td>
<td>&lt;zip2, dist2&gt;</td>
<td>&lt;zip2, dist2&gt;</td>
</tr>
<tr>
<td>&lt;60632, 874.2&gt;</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>&lt;07307, 50.94&gt;</td>
<td>{&lt;07097, 0.26&gt;, &lt;07030, 0.83&gt;, &lt;07310, 1.09&gt;, ...}</td>
<td>{}</td>
</tr>
<tr>
<td>&lt;28041, 240.46&gt;</td>
<td>{&lt;28072, 1.74&gt;, &lt;28146, 3.41&gt;, &lt;28138, 3.97&gt;, ...}</td>
<td>{&lt;28072, 1.74&gt;, &lt;28146, 3.41&gt;}</td>
</tr>
</tbody>
</table>

No Overlap

No Overlap

Overlap!
Approximating Equality

Allow flexibility in values from different sources

- **Numeric Types** like *distance*
  
  10.6 km $\approx$ 10.54 km

  Error Bounds (eg. +/- 1%)

- **Nominal Types** like *company*
  
  Google Inc. $\approx$ Google Incorporated

  String Distance Metrics (e.g. JaroWinkler Score $>$ 0.9)

- **Complex Types** like *date*
  
  Mon, 31. July 2006 $\approx$ 7/31/06

  Hand-written equality checking procedures.
Automatically build semantic models for data and services available on the larger Web

- Construct models of these sources that are sufficiently rich to support querying and integration

- Current focus:
  - Build models for the vast amount of structured and semi-structured data available
    - *Not just web services, but also form-based interfaces*
    - *E.g., Weather forecasts, flight status, stock quotes, currency converters, online stores, etc.*
  - Learn models for information-producing web sources and web services
Integrated Approach

• Start with some initial knowledge of a domain
  • Sources and semantic descriptions of those sources

• Automatically
  • Discover related sources
  • Determine how to invoke the sources
  • Learn the syntactic structure of the sources
  • Identify the semantic types of the data
  • Build semantic models of the source
  • Construct semantic web services
Seed Source
Automatically Discover and Build Semantic Web Services for Related Sources
### Background Knowledge

- **Seed URL**: http://wunderground.com

- **unisys(Zip, Temp, ...)**
  - **definition of known sources**
  - **sample values**

- **unisys(Zip, Temp, Humidity, ...)**
  - **patterns**
  - **domain types**

---

**Integrated Approach**

1. **discovery**
2. **Invocation & extraction**
3. **source modeling**
4. **semantic typing**

- **unisys**
  - **Invocation & extraction**
  - **sample input values**

- **http://wunderground.com**

- **unisys(Zip, Temp, ...)**
  - **:-weather(Zip, ..., Temp, Hi, Lo)**
Background Knowledge

- Seed URL
  - http://wunderground.com
- Sample input values
- Patterns
- Domain types
- Definition of known sources
- Sample values

unisys(Zip, Temp, ...) :- weather(Zip, ..., Temp, Hi, Lo)

unisys(Zip, Temp, Humidity, ...)

Invocation & extraction

Source modeling

Semantic typing

AnotherWS

Unisys

"90254"
Background Knowledge

- Ontology of the inputs and outputs
  - e.g., TempF, Humidity, Zipcode;
- Sample values for each semantic type
  - e.g., “88 F” for TempF, and “90292” for Zipcode
- Domain input model
  - a weather source may accept Zipcode or City and State as input
  - Sample input values
- Known sources (seeds)
  - e.g., http://wunderground.com
- Source descriptions in Datalog or RDF
  - wunderground($Z,CS,T,F0,S0,Hu0,WS0,WD0,P0,V0,FL1,FH1,S1,FL2,FH2,S2, FL3,FH3,S3,FL4,FH4,S4,FL5,FH5,S5) :-
    weather(0,Z,CS,D,T,F0,_,_,S0,Hu0,P0,WS0,WD0,V0)
    weather(1,Z,CS,D,T,_,FH1,FL1,S1,_,_,_,_,_),
    weather(2,Z,CS,D,T,_,FH2,FL2,S2,_,_,_,_,_),
    weather(3,Z,CS,D,T,_,FH3,FL3,S3,_,_,_,_,_),
    weather(4,Z,CS,D,T,_,FH4,FL4,S4,_,_,_,_,_),
    weather(5,Z,CS,D,T,_,FH5,FL5,S5,_,_,_,_,_).
Source Discovery

- **discovery**
- **unisys**
- **Invocation & extraction**
- **Background knowledge**
  - Seed URL
  - sample input values
  - http://wunderground.com
- **source modeling**
  - unisys(Zip,Temp,...)
  - definition of known sources
  - sample values
- **semantic typing**
  - patterns
  - domain types
  - unisys(Zip,Temp,Humidity,...)

**unisys(Zip,Temp,Hi,Lo)**

sample values

unisys(Zip,Temp,...)

:=-weather(Zip,...,Temp,Hi,Lo)
Leverage user-generated tags on the social bookmarking site del.icio.us to discover sources similar to the seed.
Exploiting Social Annotations for Resource Discovery

- **Resource discovery task**: "given a seed source, find other most similar sources"
  - Gather a corpus of <user, source, tag> bookmarks from del.icio.us
  - Use probabilistic modeling to find hidden topics in the corpus
  - Rank sources by similarity to the seed within topic space
Source Invocation & Extraction

- discovery
  - Seed URL
  - Background knowledge
    - Seed URL
    - Sample input values
    - Unisys (Zip, Temp, ...) :- weather (Zip, ..., Temp, Hi, Lo)
  - Definition of known sources
  - Sample values
  - Patterns
  - Domain types

- Invocation & extraction
  - Sample input values

- Source modeling
  - Unisys (Zip, Temp, Humidity, ...)

- Semantic typing
  - "90254"
To invoke the target source, we need to locate the form and determine the appropriate input values:

1. Locate the form
2. Try different data type combinations as input
   - For weather, only one input - location, which can be zipcode or city/state
3. Submit Form
4. Keep successful invocations
Inducing Extraction Templates

- **Template**: a sequence of alternating slots and *stripes*
  - stripes are the common substrings among all pages
  - slots are the placeholders for data
- **Induction**: Stripes are discovered using the Longest Common Subsequence algorithm

### Sample Page 1
- **Temp**: 72°F (22°C)
- **Site**: KSMO (Santa Monica, CA)
- **Time**: 11 AM PST 10 DEC 08

### Sample Page 2
- **Temp**: 37°F (2°C)
- **Site**: KAGC (Pittsburgh/Alle, PA)
- **Time**: 2 PM EST 10 DEC 08

---

**Template**

```
<font face="Arial, Helvetica, sans-serif">
  Temp: °C
  Site: Site Name (City, State)
  Time: 11 AM PST 10 DEC 08
</font>
```

**Slot**

```
<font face="Arial, Helvetica, sans-serif">
  (e.g., 72°F)
</font>
```

**Stripe**

```
<font face="Arial, Helvetica, sans-serif">
  (e.g., °C)
</font>
```
To extract data: Find data in slots by locating the stripes of the template on unseen page:

Unseen Page

- Temp: 71F (21C)
- Site: KCQT (Los_Angeles_Dow, CA)
- Time: 11 AM PST 10 DEC 08

Extracted Data

- Sun
- Sunny
- 71F
- 21C
- KCQT
- Los_Angeles_Dow
- CA
- 11 AM PST
Semantic Typing

- **discovery**
- **unisys**
- **Invocation & extraction**

**Background knowledge**
- Seed URL: http://wunderground.com
- Sample input values: "90254"
- Sample values: unisys(Zip, Temp, Humidity, ...)
- Definition of known sources
- Sample values

- **source modeling**
- **semantic typing**

- **patterns**
- **domain types**

unisys(Zip, Temp, Humidity, ...)

unisys(Zip, Temp, ...) :- weather(Zip, ..., Temp, Hi, Lo)
Semantic Typing
[Lerman, Plangprasopchok, & Knoblock]

✓ Idea: Learn a model of the content of data and use it to recognize new examples

![Diagram showing the process of learning patterns from background knowledge to label examples.](image-url)
Labeling New Data

• Use learned patterns to link new data to types in the ontology
  • Score how well patterns describe a set of examples
    – Number of matching patterns
    – How many tokens of the example match pattern
    – Specificity of the matched patterns
• Output top-scoring types

<table>
<thead>
<tr>
<th>Person</th>
<th>Address</th>
<th>Work</th>
</tr>
</thead>
<tbody>
<tr>
<td>E Lewis</td>
<td>3518 Hilltop Rd</td>
<td>(419) 531 - 0504</td>
</tr>
<tr>
<td>Andrew Lewis</td>
<td>3543 Larchmont Pkwy</td>
<td>(518) 474 - 4799</td>
</tr>
<tr>
<td>C. S. Lewis</td>
<td>555 Willow Run Dr</td>
<td>(612) 578 - 5555</td>
</tr>
<tr>
<td>Carmen Jones</td>
<td>355 Morgan Ave N</td>
<td>(612) 522 - 5555</td>
</tr>
<tr>
<td>John Jones</td>
<td>3574 Brookside Rd</td>
<td>(555) 531 - 9566</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Location</th>
<th>State_prov</th>
<th>Postal_code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toledo</td>
<td>OH</td>
<td>64325-3000</td>
</tr>
<tr>
<td>Toledo</td>
<td>OH</td>
<td>64356</td>
</tr>
<tr>
<td>Seattle</td>
<td>WA</td>
<td>8422</td>
</tr>
<tr>
<td>Seattle</td>
<td>WA</td>
<td>8435</td>
</tr>
<tr>
<td>Omaha</td>
<td>NE</td>
<td>52456-6444</td>
</tr>
</tbody>
</table>

:StreetAddress: :Email:
4DIG CAPS Rd   ALPHA@ALPHA.edu
3DIG N CAPS Ave ALPHA@ALPHA.com
...             ...
:State:       :Telephone:
CA             (3DIG) 3DIG-4DIG
2UPPER         +1 3DIG 2DIG 4DIG
...             ...
Source Modeling
[Carman & Knoblock]

- **discovery**
- **invocation & extraction**
- **semantic typing**

**Background knowledge**
- Seed URL
- Sample input values
- Definition of known sources
- Sample values
- Patterns
- Domain types

- **unisys**
- http://wunderground.com
- unisys(Zip, Temp, ...) :- weather(Zip, ..., Temp, Hi, Lo)
- unisys(Zip, Temp, Humidity, ...)

**Source Modeling**
Inducing Source Definitions

- Step 1: classify input & output semantic types

source1($zip, lat, long) :- centroid(zip, lat, long).

source2($lat1, $long1, $lat2, $long2, dist) :-
greatCircleDist(lat1, long1, lat2, long2, dist).

source3($dist1, dist2) :- convertKm2Mi(dist1, dist2).

source4($startZip, $endZip, separation)
Generating Plausible Definition

- **Step 1**: classify input & output semantic types
- **Step 2**: generate plausible definitions

```prolog
source1($zip, lat, long) :-
    centroid(zip, lat, long).

source2($lat1, $long1, $lat2, $long2, dist) :-
    greatCircleDist(lat1, long1, lat2, long2, dist).

source3($dist1, dist2) :-
    convertKm2Mi(dist1, dist2).

source4($zip1, $zip2, dist) :-
    source1(zip1, lat1, long1),
    source1(zip2, lat2, long2),
    source2(lat1, long1, lat2, long2, dist2),
    source3(dist2, dist).

source4($zip1, $zip2, dist) :-
    centroid(zip1, lat1, long1),
    centroid(zip2, lat2, long2),
    greatCircleDist(lat1, long1, lat2, long2, dist2),
    convertKm2Mi(dist1, dist2).
```
Invoke and Compare the Definition

- Step 1: classify input & output semantic types
- Step 2: generate plausible definitions
- Step 3: invoke service & compare output

```
source4($zip1, $zip2, dist):-
   source1(zip1, lat1, long1),
   source1(zip2, lat2, long2),
   source2(lat1, long1, lat2, long2, dist2),
   source3(dist2, dist).
```

```
source4($zip1, $zip2, dist):-
   centroid(zip1, lat1, long1),
   centroid(zip2, lat2, long2),
   greatCircleDist(lat1, long1, lat2, long2,dist2),
   convertKm2Mi(dist1, dist2).
```

<table>
<thead>
<tr>
<th>$zip1</th>
<th>$zip2</th>
<th>dist (actual)</th>
<th>dist (predicted)</th>
</tr>
</thead>
<tbody>
<tr>
<td>80210</td>
<td>90266</td>
<td>842.37</td>
<td>843.65</td>
</tr>
<tr>
<td>60601</td>
<td>15201</td>
<td>410.31</td>
<td>410.83</td>
</tr>
<tr>
<td>10005</td>
<td>35555</td>
<td>899.50</td>
<td>899.21</td>
</tr>
</tbody>
</table>
wunderground( $Z, CS, T, F0, C0, S0, Hu0, WS0, WD0, P0, V0, FL1, FH1, S1, FL2, FH2, S2, FL3, FH3, S3, FL4, FH4, S4, FL5, FH5, S5):-

Weather(_w0), hasForecastDay(_w0, 0), hasZIP(_w0, Z),
hasCityState(_w0, CS), hasTimeWZone(_w0, T),
hasCurrentTemperatureFarenheit(_w0, F0),
hasCurrentTemperatureCentigrade(_w0, C0),
hasSkyConditions(_w0, S0), hasHumidity(_w0, Hu0),
hasPressure(_w0, P0), hasWindSpeed(_w0, _ws1),
WindSpeed(_ws1), hasWindSpeedInMPH(_ws1, WS0),
hasWindDir(_ws1, WD0), hasVisibilityInMi(_w0, V0),
Weather(_w1), hasForecastDay(_w1, 1), hasZIP(_w1, Z),
hasCityState(_w1, CS), hasLowTemperatureFarenheit(_w1, FL1),
hasHighTemperatureFarenheit(_w1, FH1), hasSkyConditions(_w1, S1),
...

convertC2F($C, F) :- centigrade2farenheit(C, F)
unisys($Z,_,_,_,_,_,_,_,F9,_,C,_,F13,F14,Hu,_,F17,_,_,_,_,S22,_,S24,  
_,_,_,_,_,_,_,_,_,_,S35,S36,_,_,_,_,_,_,_,_,_):-  
  wunderground(Z,_,_,F9,_,Hu,_,_,_,_,F14,F17,S24,_,_,S22,_,_,  
  S35,_,_,S36,F13,_,_),  
  convertC2F(C,F9)
Experimental Evaluation

- Experiments in 5 domains
  - Flight – lookup the current status of a flight
  - Geospatial – map street addresses into lat/long coordinates
  - Weather – find the current and forecasted weather
  - Currency – convert between various currencies
  - Mutual Funds – look up current data on a mutual fund

- Evaluation:
  - 1) Can the system correctly learn a model for those sources that perform the same task
  - 2) What is the precision and recall of the attributes in the model
### Evaluation of the Models

<table>
<thead>
<tr>
<th>domain</th>
<th>Precision</th>
<th>Recall</th>
<th>F$_1$-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>weather</td>
<td>0.64</td>
<td>0.29</td>
<td>0.39</td>
</tr>
<tr>
<td>geospatial</td>
<td>1.00</td>
<td>0.86</td>
<td>0.92</td>
</tr>
<tr>
<td>flights</td>
<td>0.69</td>
<td>0.35</td>
<td>0.46</td>
</tr>
<tr>
<td>currency</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>mutualfund</td>
<td>0.72</td>
<td>0.30</td>
<td>0.42</td>
</tr>
</tbody>
</table>
Conclusions

• Integrated approach to learning:
  • *How to invoke a web service*
  • *The semantic types of the output*
  • *A definition of what the service does*

• Provides an approach to generate source descriptions for the Semantic Web
  • Little motivation for providers to annotate services
  • Instead we generate metadata automatically

• Also provides an approach to automatically discover new sources of data