Learning Semantic Descriptions of Web Information Sources

Mark James Carman*
Information Sciences Institute / USC
&
ITC-irst / University of Trento

Craig A. Knoblock
Information Sciences Institute / USC

* I am currently seeking a Postdoc position in Europe somewhere near northern Italy ...
Mediators & Source Definitions

- Explosion of online information sources
- Mediators run queries over multiple sources
- Require declarative source definitions
- New service $\rightarrow$ model it automatically?

**Query**

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

**Source Definitions:**
- Orbitz Flight Search
- KLM Online
- Qantas Specials

**Generate Model of Service?**

```
lowestFare("MXP","HYD")
calcPrice("MXP","HYD","economy")
```

**Motivation**

Approach      Search      Scoring      Experiments      Related Work      Conclusions
Modeling Sources: an Example

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).

Step 1:
classify input & output semantic types, using:
- Metadata (labels)
- Data (content)

source4( $startZip, $endZip, separation)
Modeling Sources: Step 2

Step 2:
model functionality by:
• generating 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) :-
  centroid(zip1, lat1, long1),
  centroid(zip2, lat2, long2),
  greatCircleDist(lat1, long1, lat2, long2, dist2),
  convertKm2Mi(dist1, dist2).

source1(zip1, lat1, long1),
source1(zip2, lat2, long2),
source2(lat1, long1, lat2, long2, dist2),
source3(dist2, dist).
Modeling Sources: Step 2

Step 2:

model functionality by:

- generating plausible definitions
- comparing the output they produce

source4( $zip1, $zip2, dist) :-

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

<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>
Summary - Modeling Sources

Step 1: Semantic Labeling
Classify input & output *semantic types*, using:
- Labels: metadata
- Content: output data

Step 2: Functional Modeling
Model the *functionality* of service by:
- Search: generating plausible definitions
- Scoring: compare the output they produce
Summary - Modeling Sources

Step 1: Semantic Labeling

Chose sources, using:
- Lerman, Plangprasopchok and Knoblock.
- Automatically labeling data used by web services.
- AAAI’06.

Previous Work!

Step 2: Functional Modeling

Model the *functionality* of service by:
- Search: generating plausible definitions
- Scoring: compare the output they produce

---

<table>
<thead>
<tr>
<th>Example Inputs</th>
<th>New Source</th>
<th>Known Source</th>
<th>Candidate Tuples</th>
<th>Target Tuples</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Invoke Source</td>
<td>Execute definition</td>
<td>Compare outputs</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Searching for Definitions

- Search space of conjunctive queries:
  \[ \text{target}(X) \text{ :- source1}(X_1), \text{source2}(X_2), \ldots \]

Invoke \text{target} with set of random inputs;
Add empty clause to \text{queue};

while (\text{queue} not empty)
  \[ \nu := \text{best definition from queue} \; ; \]
  \[ \text{forall} \; (\nu' \text{ in Expand}(\nu)) \]
  \[ \text{if} \; (\text{Eval}(\nu') > \text{Eval}(\nu)) \]
  \[ \text{insert } \nu' \text{ into queue} \; ; \]

1. Sample the new source

Expressive Language
Sufficient for modeling most online sources

2. Best-first search through space of candidate definitions
Invoking the Target

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

Invoke source with randomly generated tuples
- Use distribution if available
- If no output is produced try invoking other sources

<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;,</td>
</tr>
<tr>
<td></td>
<td>&lt;07030, 0.83&gt;,</td>
</tr>
<tr>
<td></td>
<td>&lt;07310, 1.09&gt;, ...}</td>
</tr>
<tr>
<td>&lt;60632, 10874.2</td>
<td>{}</td>
</tr>
</tbody>
</table>

randomly generated input tuples

Empty Result

Non-empty Result
Top-down Generation of Candidates

Start with empty clause & specialize it by:

- Adding a predicate from set of sources
- Check that definition is not redundant

```
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)
```

Expand
Best-first Enumeration of Candidates

Evaluate clauses & expand the best one

\[ \text{source5}(\_,\_,\_,\_). \]

\[ \text{source5}(\text{zip1},\_,\_,\_). \]
\[ \text{source4}(\text{zip1},\text{zip1},\_). \]

\[ \text{source5}(\text{zip1},\_,\text{zip2},\text{dist2}). \]
\[ \text{source4}(\text{zip2},\text{zip1},\text{dist2}). \]

\[ \text{source5}(\_,\text{dist1},\_,\text{dist2}). \]
\[ <(\text{dist2},\text{dist1}). \]

\[ \text{source5}(\text{zip1},\text{dist1},\text{zip2},\text{dist2}). \]
\[ \text{source4}(\text{zip2},\text{zip1},\text{dist2}). \]
\[ \text{source4}(\text{zip1},\text{zip2},\text{dist1}). \]
\[ <(\text{dist2},\text{dist1}). \]

\[ \text{source5}(\text{zip1},\text{dist1},\text{zip2},\text{dist2}). \]
\[ \text{source4}(\text{zip2},\text{zip1},\text{dist2}). \]
\[ <(\text{dist2},\text{dist1}). \]

...
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

Standard techniques
Non-standard technique
**Scoring Candidates**

Need to score candidates to direct best-first search
- Score definitions based on overlap

<table>
<thead>
<tr>
<th><strong>Input</strong></th>
<th><strong>Target Output</strong></th>
<th><strong>Clause Output</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;60632, 874.2&gt;</td>
<td>{}</td>
<td>{{&lt;60629, 2.15&gt;, &lt;60682, 2.27&gt;, &lt;60623, 2.64&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>
<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>

---

**Motivation**
- Approach
- Search
- Scoring
- Experiments
- Related Work
- Conclusions
Scoring Candidates II

Sources may return multiple tuples and not be complete:
- Use Jaccard similarity as fitness function
- Average results across different inputs

\[
\text{return } \text{average}(\text{fitness}) \\
\text{for all (tuplen in InputTuples)} \\
T_{\text{target}} = \text{invoke}(\text{target, tuple}) \\
T_{\text{clause}} = \text{execute}(\text{clause, tuple}) \\
\text{if not } (|T_{\text{target}}|=0 \text{ and } |T_{\text{clause}}|=0) \\
\text{fitness} = \frac{|T_{\text{target}} \cap T_{\text{clause}}|}{|T_{\text{target}} \cup T_{\text{clause}}|} \\
\text{return average(fitness)}
\]

At least half of input tuples are non-empty invocations of target

Average results only when output is returned

Jaccard similarity

Motivation      Approach     Search     Scoring     Experiments     Related Work     Conclusions
Approximating Equality

Allow flexibility in values from different sources

- Numeric Types like *distance*
  
  \[10.6 \text{ km} \approx 10.54 \text{ km}\]
  Error Bounds (e.g. +/- 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.
Experimental Setup

- 25 problems
- 35 known sources
- All real services
- Time limit of 20 minutes

Equality Approximations:
- 1% for distance, speed, temperature & price
- 0.002 degrees for latitude & longitude
- JaroWinkler > 0.85 for company, hotel & airport
- Hand-written procedure for date.

Inductive search bias:
- Max clause length: 7
- Predicate repetition: 2
- Max variable level: 5
- Executable candidates
- No variable repetition
Actual Learned Examples

1. `GetDistanceBetweenZipCodes($zip0, $zip1, dis2):- GetCentroid(zip0, lat1, lon2), GetCentroid(zip1, lat4, lon5), GetDistance(lat1, lon2, lat4, lon5, dis10), ConvertKm2Mi(dis10, dis2).`

2. `USGSElevation($lat0, $lon1, dis2):- ConvertFt2M(dis2, dis1), Altitude(lat0, lon1, dis1).`

3. `YahooWeather($zip0, cit1, sta2, , lat4, lon5, day6, dat7,tem8, tem9, sky10) :- WeatherForecast(cit1,sta2,,lat4,lon5,,day6,dat7,tem9,tem8,,sky10,,), GetCityState(zip0, cit1, sta2).`

4. `GetQuote($tic0,pri1,dat2,tm3,pri4,pri5,pri6,pri7,cou8,,pri10,,pri13,,com15) :- YahooFinance(tic0, pri1, dat2, tim3, pri4, pri5, pri6,pri7, cou8), GetCompanyName(tic0,com15,,),Add(pri5,pri13,pri10),Add(pri4,pri10,pri1).`

5. `YahooAutos($zip0, $mak1, dat2, yea3, mod4, , , pri7, ) :- GoogleBaseCars(zip0, mak1, , mod4, pri7, , , yea3), ConvertTime(dat2, , dat10, , ), GetCurrentTime( , , dat10, ).`
Experimental Results

Overall Results:
- Average Precision: 88%
- Average Recall: 69%

Results for different domains:

<table>
<thead>
<tr>
<th>Problem Domain</th>
<th># of Problems</th>
<th>Avg. # of Candidates</th>
<th>Avg. Time (s)</th>
<th>Avg. Precision</th>
<th>Avg. Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geospatial</td>
<td>9</td>
<td>136</td>
<td>303</td>
<td>100%</td>
<td>84%</td>
</tr>
<tr>
<td>Financial</td>
<td>2</td>
<td>1606</td>
<td>335</td>
<td>56%</td>
<td>63%</td>
</tr>
<tr>
<td>Weather</td>
<td>8</td>
<td>368</td>
<td>693</td>
<td>91%</td>
<td>62%</td>
</tr>
<tr>
<td>Hotels</td>
<td>4</td>
<td>43</td>
<td>374</td>
<td>90%</td>
<td>60%</td>
</tr>
<tr>
<td>Cars</td>
<td>2</td>
<td>68</td>
<td>940</td>
<td>50%</td>
<td>50%</td>
</tr>
</tbody>
</table>
Related Work

Semantic Labeling:
- Metadata-based service classification (Hess & Kushmerick, ’03)
- Woogle: Web Service clustering (Dong et al, 2004)
  - Neither system produces sufficient information for integration

Functional Modeling:
- Category Translation (Perkowitz & Etzioni 1995)
  - Less complicated (single input, single output) definitions.
- iMAP: Complex schema matcher (Dhamanka et. al. 2004)
  - Many-to-1 not many-to-many mappings
  - Type-specific search algorithms
  - Not designed for live information sources
Conclusions

- Assumption: overlap between new & known sources
- Technique is nonetheless widely applicable:
  - Redundancy
  - Scope or Completeness
  - Binding Constraints
  - Composed Functionality
  - Access Time
Conclusions

- Integrated approach for learning:
  - *How to invoke a web service (inputs & outputs)*
  - *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

- Provides approach to discover new sources of data automatically