Doctoral Thesis:
Learning Semantic Definitions for Information Sources on the Internet

Mark James Carman

Advisors:
Prof. Paolo Traverso
Prof. Craig A. Knoblock
Abundance of Information Sources

Motivation
Approach
Search
Scoring
Extensions
Experiments
Related Work
Conclusions
Bringing the Data Together
Bringing the Data Together
Mediators resolve Heterogeneity

**Motivation**

**Approach**

**Search**

**Scoring**

**Extensions**

**Experiments**

**Related Work**

**Conclusions**
Mediators Require Source Definitions

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

```
SELECT MIN(price)
FROM flight
WHERE depart="LAX"
AND arrive="MXP"
```
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).
  
  source4( $startZip, $endZip, separation) :-
  Assumed this problem has been solved!
**Inducing Source Definitions - Step 2**

- **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).
```

Known Source 1

Known Source 2

Known Source 3
Inducing Source Definitions – Step 3

- **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>

Match
Overlapping Data Requirement

- Assumption: overlap between new & known sources
- Nonetheless, the technique is widely applicable:
  - Redundancy
  - Scope or Completeness
  - Binding Constraints
  - Composed Functionality
  - Access Time
Searching for Definitions

- Search space of *conjunctive queries*:
  \[ \text{target}(X) :- \text{source}1(X_1), \text{source}2(X_2), \ldots \]
- For scalability don’t allow negation or union
- Perform Top-Down Best-First Search

1. First sample the New Source
2. Then perform best-first search through space of candidate definitions

For scalability don’t allow negation or union

Expressive Language Sufficient for modeling most online sources
Invoking the Target

Invoke source with \textit{representative} values

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

\begin{tabular}{|l|l|}
\hline
\textbf{Input} & \textbf{Output} \\
\hline
<zip1, dist1> & \{<07097, 0.26>, <07030, 0.83>, <07310, 1.09>, \ldots\} \\
<60632, 10874.2> & \{\} \\
\hline
\end{tabular}
Invoking the Target

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).
```

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

---

**New Source 5**
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
  - Standard ILP techniques
  - Non-standard technique
Evaluating Candidates

- 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>{&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    Extensions    Experiments    Related Work    Conclusions
Evaluating Candidates III

PROBLEM: All sources assumed incomplete
- Even *optimal definition* may only produce overlap
- Want definition that *best predicts* the target’s output
- Use Jaccard similarity to score candidates

Motivation    Approach    Search    Scoring    Extensions    Experiments    Related Work    Conclusions

forall (tuple in InputTuples)

\[
\begin{align*}
T_{\text{target}} &= \text{invoke}(\text{target, tuple}) \\
T_{\text{clause}} &= \text{execute}(\text{clause, tuple}) \\
\text{if not } (|T_{\text{target}}|=0 \text{ and } |T_{\text{clause}}|=0)
\end{align*}
\]

\[
\text{fitness} = \frac{|T_{\text{target}} \cap T_{\text{clause}}|}{|T_{\text{target}} \cup T_{\text{clause}}|}
\]

return average(fitness)

At least half of input tuples are non-empty invocations of target
Similarity metric is Jaccard similarity between the sets
Average results only when output is returned
Missing Output Attributes

- Some candidates produce less output attributes:
  - Makes comparing them difficult

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

- Penalize candidate by number of “negative examples”

- First candidate doesn’t produce either outputs, thus:
  - Penalty = | {zipcode}| x | {distance}|  
  - For numeric types use accuracy to approximate cardinality
Different Input Attributes

- Some clauses take different inputs from target:
  
  ```prolog
  source5($zip1,$dist1,zip2,_) :- source4($zip1,$zip2,dist1).
  ```

- `zip2` is an input parameter for clause but not target.
- Should invoke operation with *every possible zip code!*
  
  > > 40,000 zip codes in US

- Problem: algorithm should return & not get banned!
- Solution: sample to estimate score for clause:
  
  - record the scaling factor = | {zipcode}/ #invocations
  - bias search: choose at least half of tuples to be positive
Approximating Equality

Allow flexibility in values from different sources

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

  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.
Extensions

Many extensions to basic algorithm are discussed in thesis:

- Inverse and functional sources
- Constants in the modeling language
- Post-processing (tightening) of definitions
- Search heuristics based on semantic types
- Caching & determining if source is blocking
Experiments – Setup

Problems:
- 25 target predicates
- *same* domain model
  (70 Semantic Types and 37 Predicates)
- 35 known sources

System Settings:
- Each target source invoked at least 20 times
- Time limit of 20 minutes imposed

Inductive search bias:
- Maximum clause length 7
- Predicate repetition limit 2
- Maximum variable level 5
- Candidate must be executable
- Only 1 variable occurrence per literal

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*.
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, tim3, 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

Results for different domains:

<table>
<thead>
<tr>
<th>Problem Domain</th>
<th># of Problems</th>
<th>Avg. # of Candidates</th>
<th>Avg. Time (sec)</th>
<th>Attributes Learnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>geospatial</td>
<td>9</td>
<td>136</td>
<td>303</td>
<td>84%</td>
</tr>
<tr>
<td>financial</td>
<td>2</td>
<td>1606</td>
<td>335</td>
<td>59%</td>
</tr>
<tr>
<td>weather</td>
<td>7</td>
<td>368</td>
<td>693</td>
<td>69%</td>
</tr>
<tr>
<td>hotels</td>
<td>4</td>
<td>43</td>
<td>374</td>
<td>60%</td>
</tr>
<tr>
<td>cars</td>
<td>2</td>
<td>68</td>
<td>940</td>
<td>50%</td>
</tr>
</tbody>
</table>
Comparison with Other Systems

ILA & Category Translation (Perkowitz & Etzioni 1995)
Learn functions describing operations on internet
- My system learns *more complicated* definitions
  - Multiple attributes, Multiple output tuples, etc.

iMAP (Dhamanka et. al. 2004)
Discovers complex (*many-to-1*) mappings between DB schemas
- My system learns *many-to-many* mappings
- My approach is more general (single search algorithm)
- Deal with problem of invoking sources
Conclusions

Learning procedure for online information services is:

1. **Automated**
2. **Expressive** *(conjunctive queries)*
3. **Efficient** *(access sources only as required)*
4. **Robust** *(to noisy and incomplete data)*
5. **Evolving** *(improves with # of known sources)*
6. **Scalable** *(for moderate size domain model)*

Generate Semantic Metadata for Semantic Web

- Little motivation for providers to annotate services
- Instead we generate metadata automatically