Doctoral Thesis:

Learning Semantic Definitions for Information Sources on the Internet

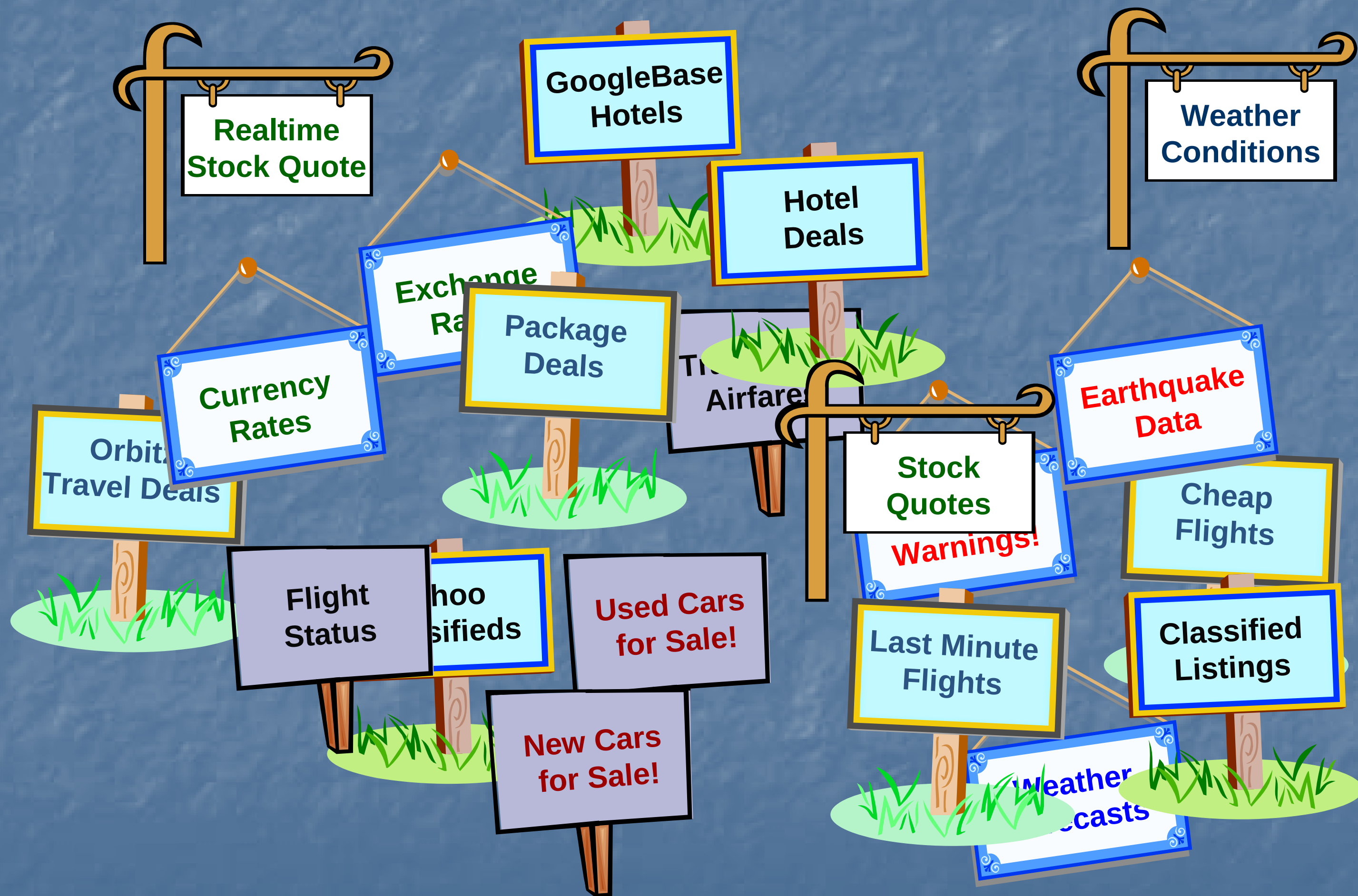
Mark James Carman

Advisors:

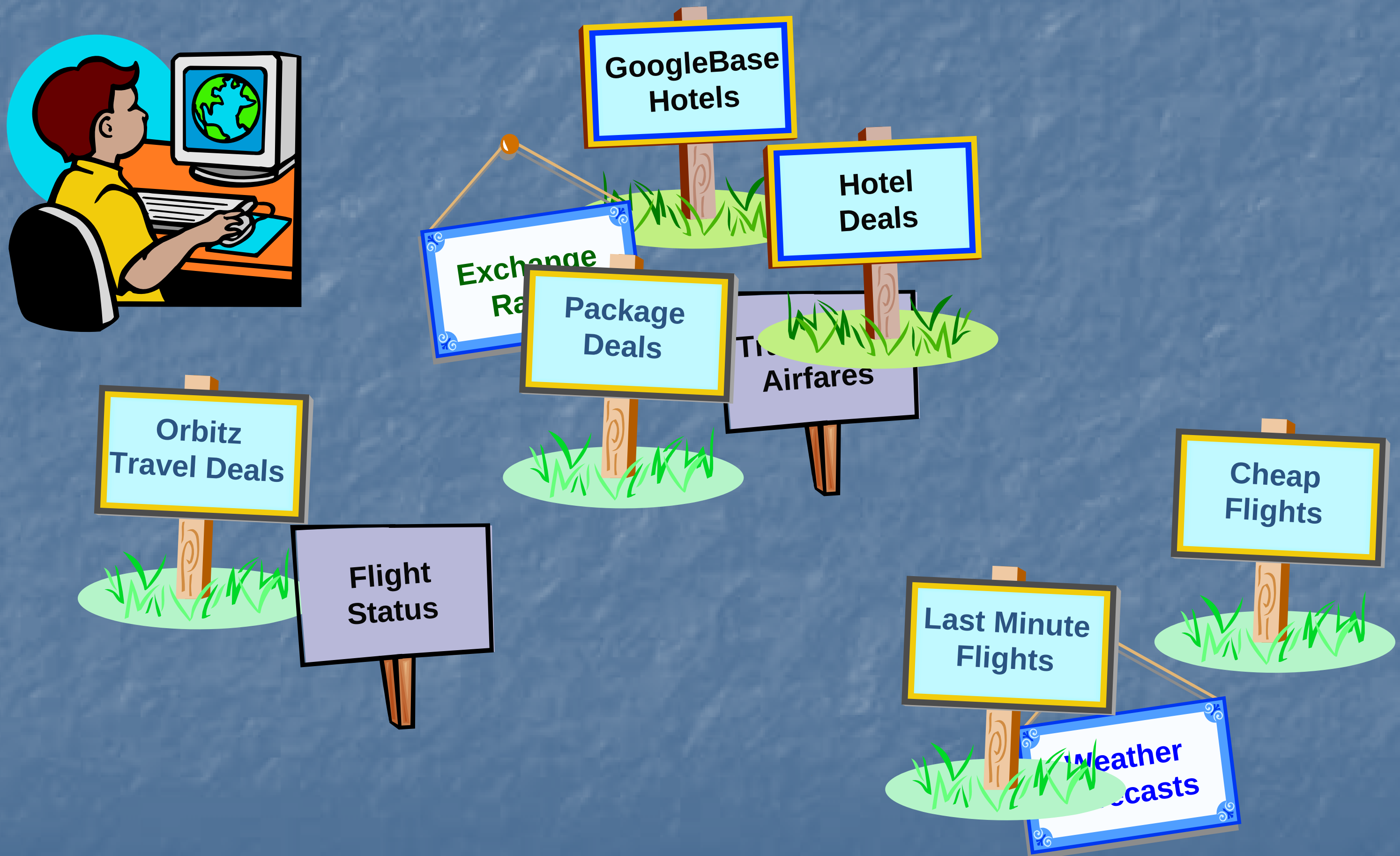
Prof. Paolo Traverso

Prof. Craig A. Knoblock

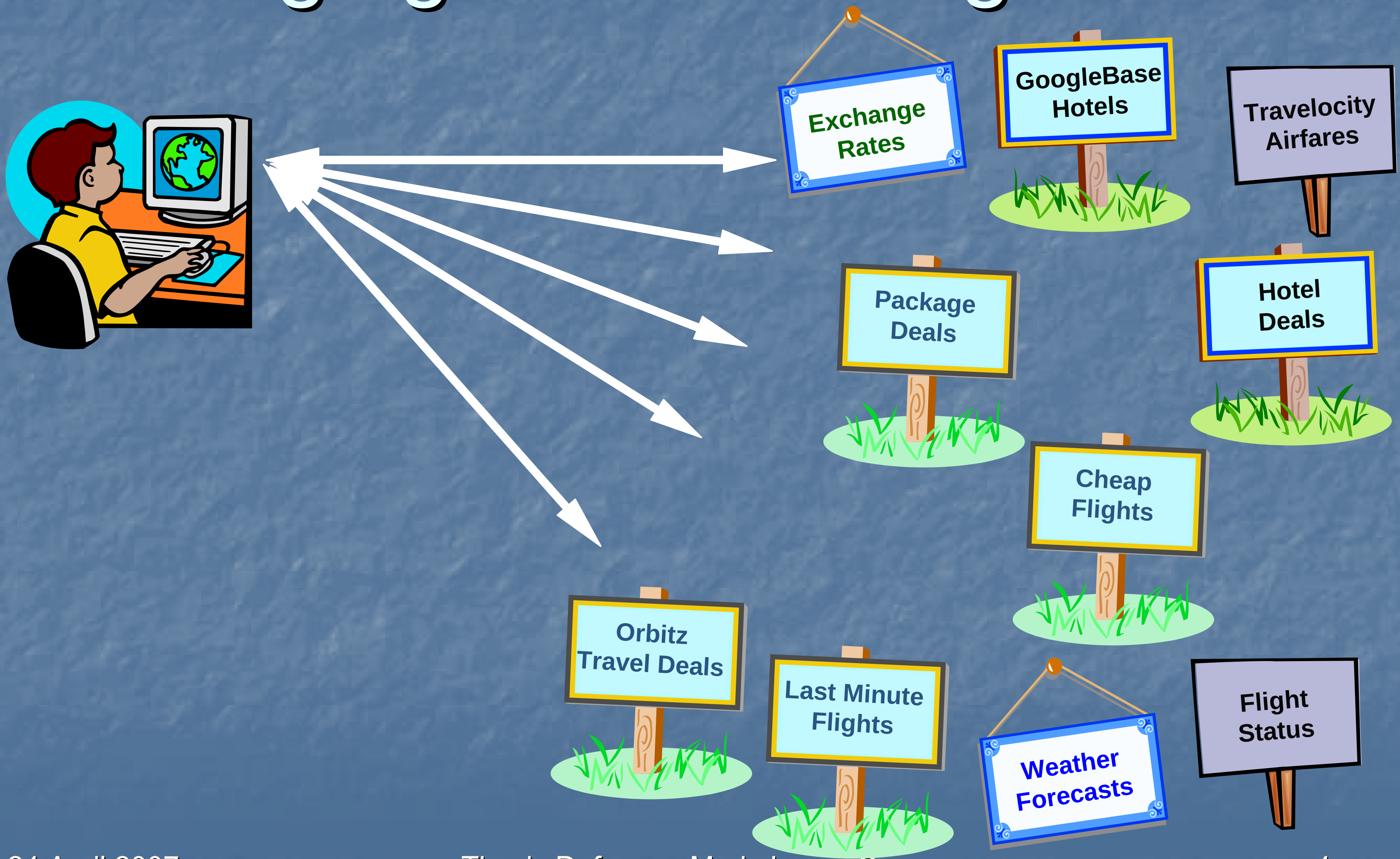
Abundance of Information Sources



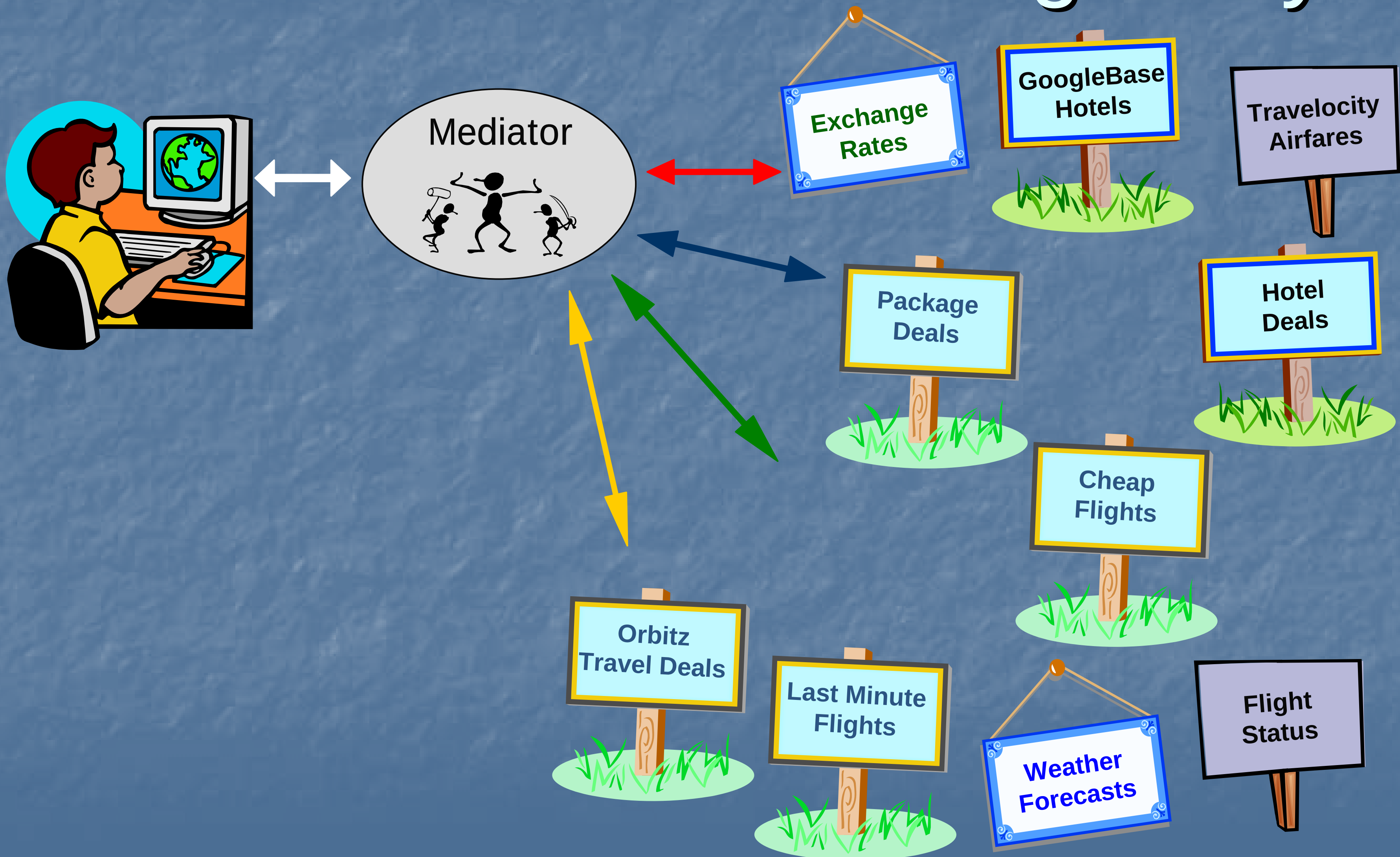
Bringing the Data Together



Bringing the Data Together

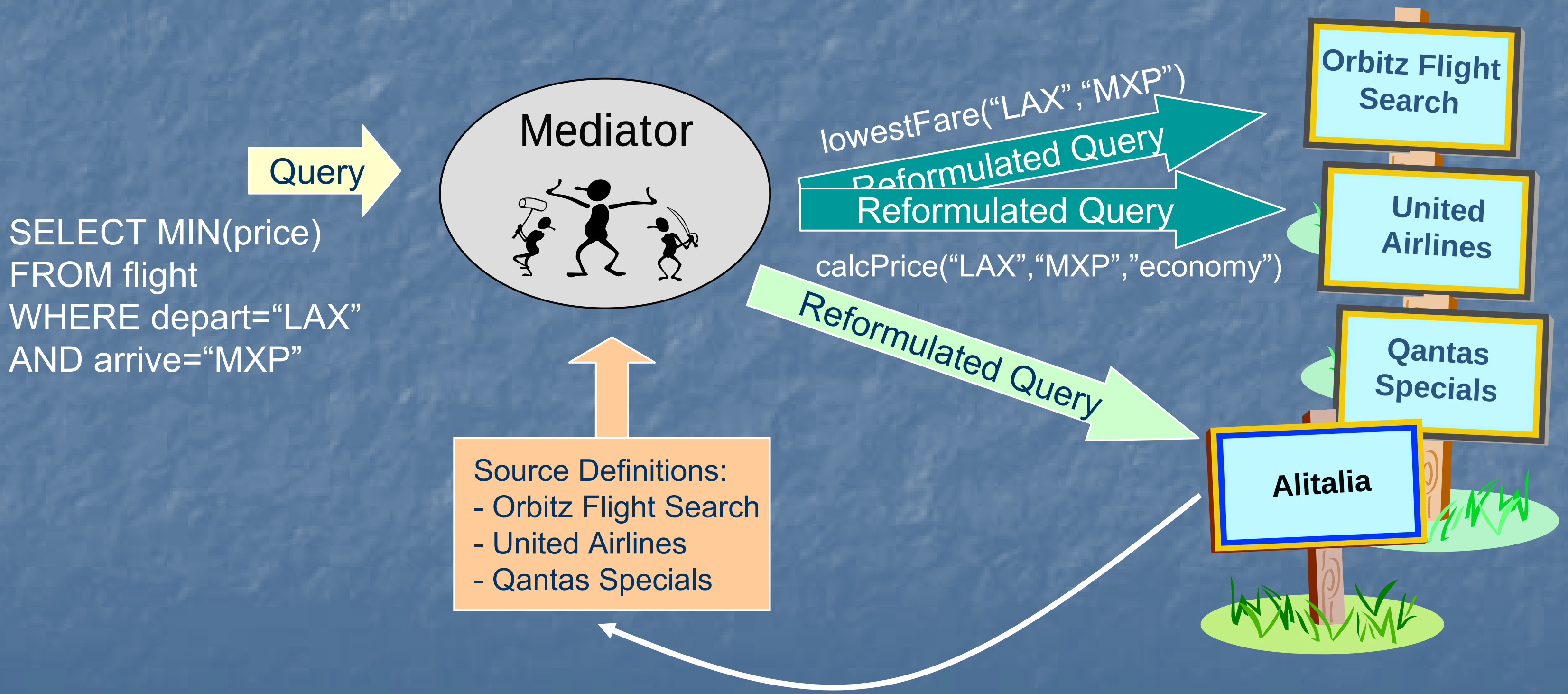


Mediators resolve Heterogeneity

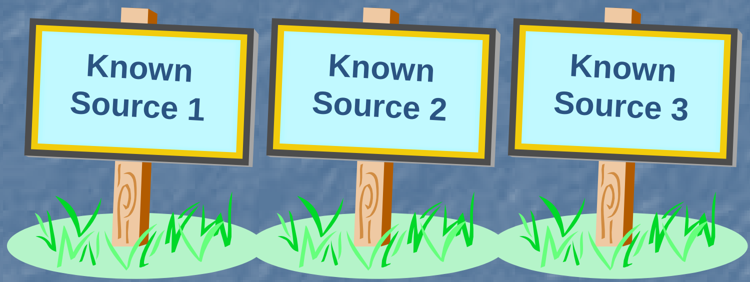


Mediators Require Source Definitions

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



Inducing Source Definitions by Example

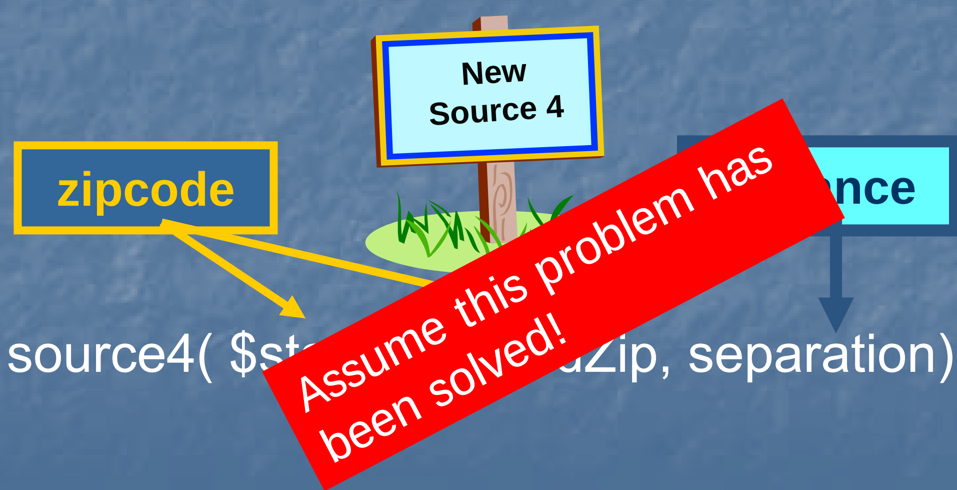


source1($zip, lat, long) :-
  centroid(zip, lat, long).

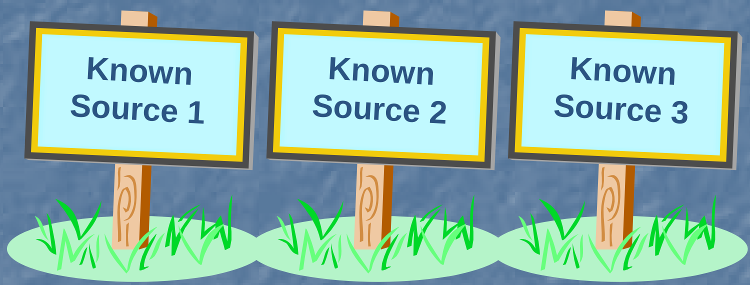
source2($lat1, $long1, $lat2, $long2, dist) :-
  greatCircleDist(lat1, long1, lat2, long2, dist).

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

- Step 1: classify input & output semantic types



Inducing Source Definitions - Step 2



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
- Step 2: generate plausible definitions

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).

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).
```

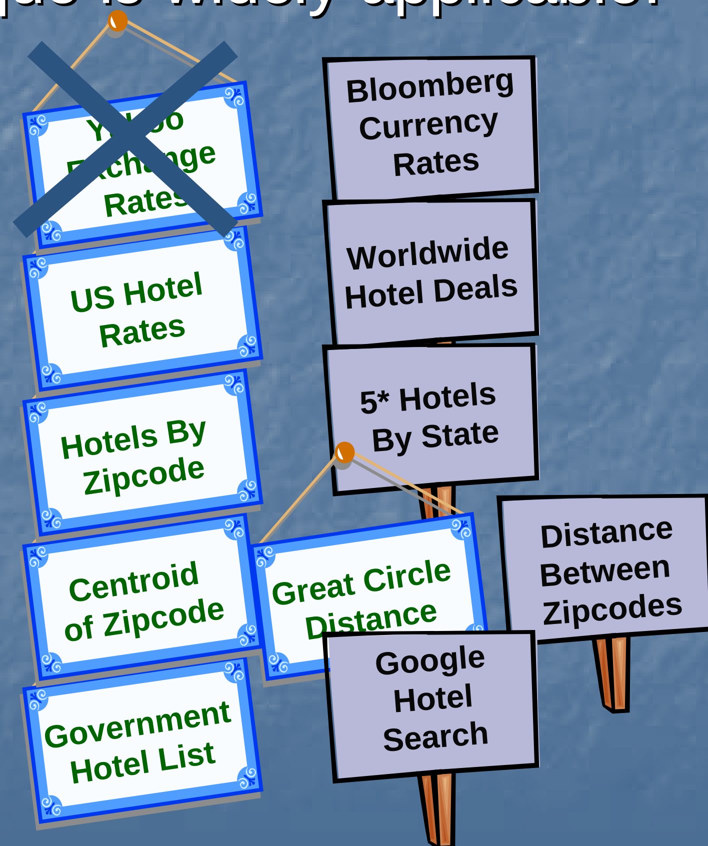
```
source4($zip1, $zip2, dist):-
    centroid(zip1, lat1, long1),
    centroid(zip2, lat2, long2),
    greatCircleDist(lat1, long1, lat2, long2, dist2),
    convertKm2Mi(dist1, dist2).
```

match

$zip1	$zip2	dist *(actual)*	dist *(predicted)*
80210	90266	842.37	843.65
60601	15201	410.31	410.83
10005	35555	899.50	899.21

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:*

  target(X) :- source1(X_1), source2(X_2), ...

Expressive Language
Sufficient for modeling most online sources

- 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***;
  forall (***v'*** in **Expand**(***v***))
    if (**Eval**(***v'***) > **Eval**(***v***))
      insert ***v'*** into ***queue***;

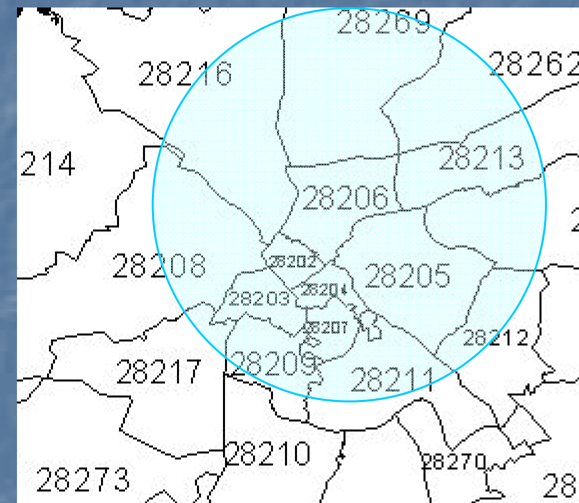
2. Then perform best-first search through space of candidate definitions

Invoking the Target

Generate Input Tuples: *<zip1, dist1>*

Invoke

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



Invoke source with *representative* values

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

Input <zip1, dist1>	**Output** <zip2, dist2>
<07307, 50.94>	{<07097, 0.26>, <07030, 0.83>, <07310, 1.09>, ...}
<60632, 10874.2>	{}

Randomly Combined Example Values

Non-empty Result

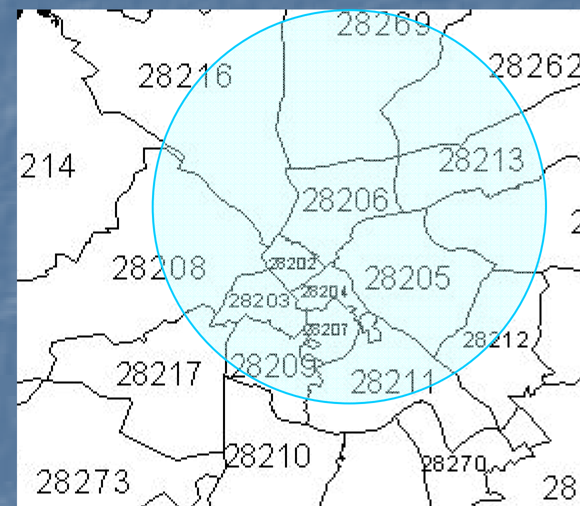
Empty Result

Invoking the Target

Generate Input Tuples: *<zip1, dist1>*

Invoke

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



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($zip1,$dist1,zip2,dist2)

source5(_,_,_,_).

Expand

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

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).

Expand

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

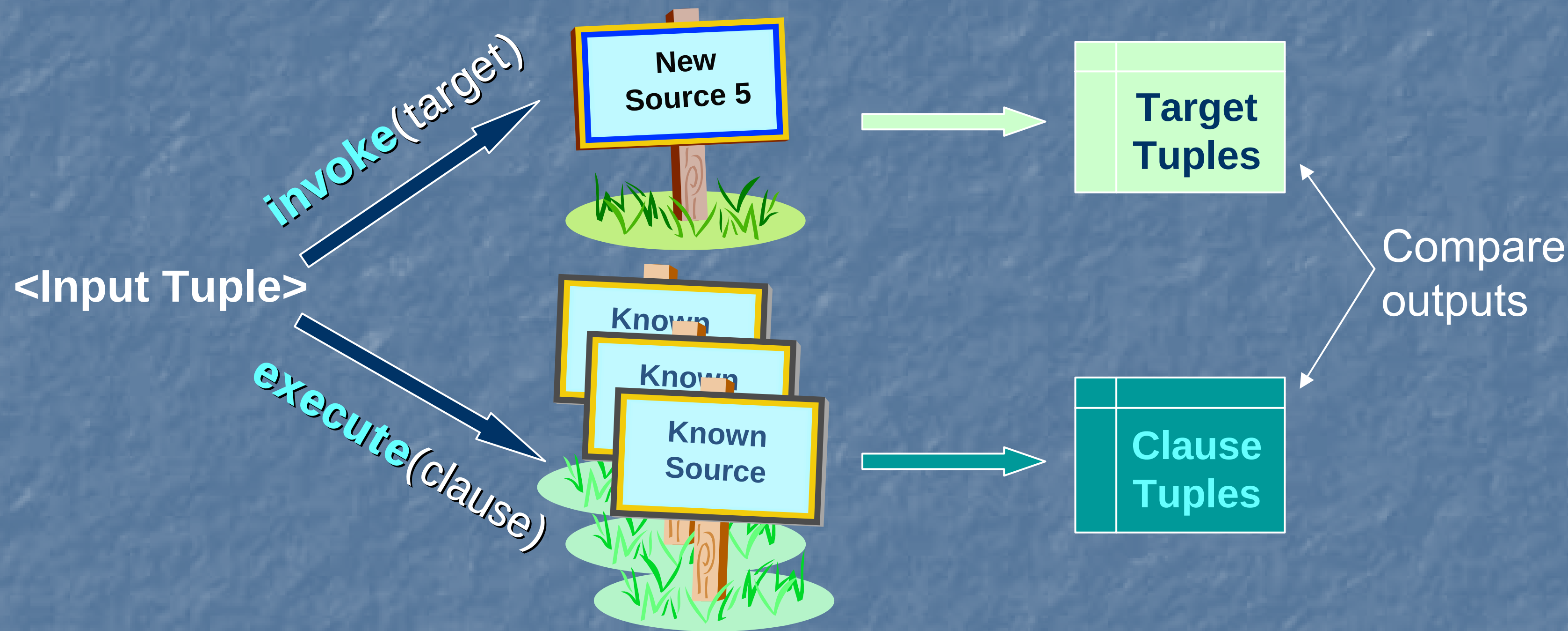
Limiting the Search

- 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!

Input <$zip1, $dist1>	**Target Output** <zip2, dist2>	**Clause Output** <zip2, dist2>
<60632, 874.2>	{}	{<60629, 2.15>, <60682, 2.27>, <60623, 2.64>, ..}
<07307, 50.94>	{<07097, 0.26>, <07030, 0.83>, <07310, 1.09>, ...}	{}
<28041, 240.46>	{<28072, 1.74>, <28146, 3.41>, <28138, 3.97>,...}	{<28072, 1.74>, <28146, 3.41>}

No Overlap

No Overlap

Overlap!

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

```
forall (tuple in InputTuples)
        T_target = invoke(target, tuple)
        T_clause = execute(clause, tuple)
        if not (|T_target|=0 and |T_clause|=0)
```

$$fitness = \frac{|T\_target \cap T\_clause|}{|T\_target \cup T\_clause|}$$

```
return average(fitness)
```

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

Average results only when output is returned

Similarity metric is Jaccard similarity between the sets

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”

source5($zipcode, $distance, **zipcode**, **distance**)

- 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:

source5(**$zip1**,**$dist1**,zip2,_) :- **source4**(**$zip1**,**$zip2**,dist1).

Target Input — **Clause Input**

- 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 km ≈ 10.54 km

  Error Bounds (eg. +/- 1%)

- Nominal Types like *company*

  Google Inc. ≈ Google Incorporated

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

- Complex Types like *date*

  Mon, 31. July 2006 ≈ 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 predicate
- *same* domain mode
  (70 Semantic Types a
- 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).

Distinguished forecast from current conditions

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).

current price = yesterday's close + change

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:

Problem Domain	# of Problems	Avg. # of Candidates	Avg. Time (sec)	Attributes Learnt
geospatial	9	136	303	84%
financial	2	1606	335	59%
weather	7	368	693	69%
hotels	4	43	374	60%
cars	2	68	940	50%

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