I — *Automatch*: Database Schema Matching Using Machine Learning with Feature Selection[1]

II — *TupleRank*: Ranking Discovered Content in Virtual Databases[2]

Jacob Berlin and Amihai Motro

1. *Proceedings of CoopIS 2001, Sixth IFCIS International Conference on Cooperative Information Systems*, pages 108-122.
2. *Proceedings of NGITS 2006, Sixth International Conference on Next Generation Information Technology and Systems, pages 13-25*

I — Automatch: Schema Matching

- Schema matching: The problem of finding mappings between the attributes of two semantically related database schemas.
- An important, current issue for many database applications:
  - Schema integration, data warehousing, electronic commerce.
- Remains largely a manual, labor-intensive process.
- Consequently, database applications that require schema matching are limited to environments in which the set of member sources is *small* and *stable*.
- These applications would scale-up to much larger communities of member sources, if the schema matching "bottleneck" was broken by automating the matching process.
- *Automatch*: A schema matching tool based on Autoplex technology.

Overall Approach

- *Attribute dictionary*: A knowledge base on schema attributes constructed from examples.
- Consider two "client" schemas that need to be matched.
  - Their corresponding *instances* are also available.
- Check every client attribute against the dictionary:
  - Obtain a matching *score* for each client attribute and each dictionary attribute.
- Combine these *client-dictionary* attribute scores to *client-client* attribute scores:
  - Assume B attribute of scheme 1, C attribute of scheme 2, A attribute of dictionary.
  - Assume B$\leftrightarrow$A is scored w_1, C$\leftrightarrow$A is scored w_2.
  - Then $B \leftrightarrow C$ is scored w_1+w_2.
- Combine these client-client attribute scores to *schema-schem*a matching scores:
  - Assume $R_1=\{B_1,B_2\}$, $R_2=\{C_1,C_2\}$.
  - w_1: $B_1 \leftrightarrow C_2$, w_2: $B_1 \leftrightarrow C_2$, w_3: $B_2 \leftrightarrow C_1$, w_4: $B_2 \leftrightarrow C_2$.
  - Then the schema matching $\{B_1 \leftrightarrow C_2, B_2 \leftrightarrow C_1\}$ is scored w_2+w_3.
- Adopt the schema matching with highest score.

Formal Statement of Problem

- *Database schema*: finite set of attributes $\{A_1, \ldots, A_n\}$.
- Given two database schemas $R_1=\{B_1, \ldots, B_p\}$ and $R_2=\{C_1, \ldots, C_q\}$
- A *matching* is a mapping between a subset of R_1 and a subset of R_2.
- Assume an *attribute dictionary D*, in which each attribute is characterized by a (select) set of possible values and their probability estimates.
- Assume a scoring function *f* that, given
  - An attribute dictionary *D*,
  - Two schemas R_1 and R_2,
  - Two corresponding instances r_1 and r_2,
  - A matching between R_1 and R_2.

  issues a score (real value) that indicates the "goodness" of the matching.
- The problem: Find the best matching of R_1 and R_2.
- We must discuss
  - The nature of *D* and *f*.
  - How to optimize *f* (find the best schema matching).

The Attribute Dictionary

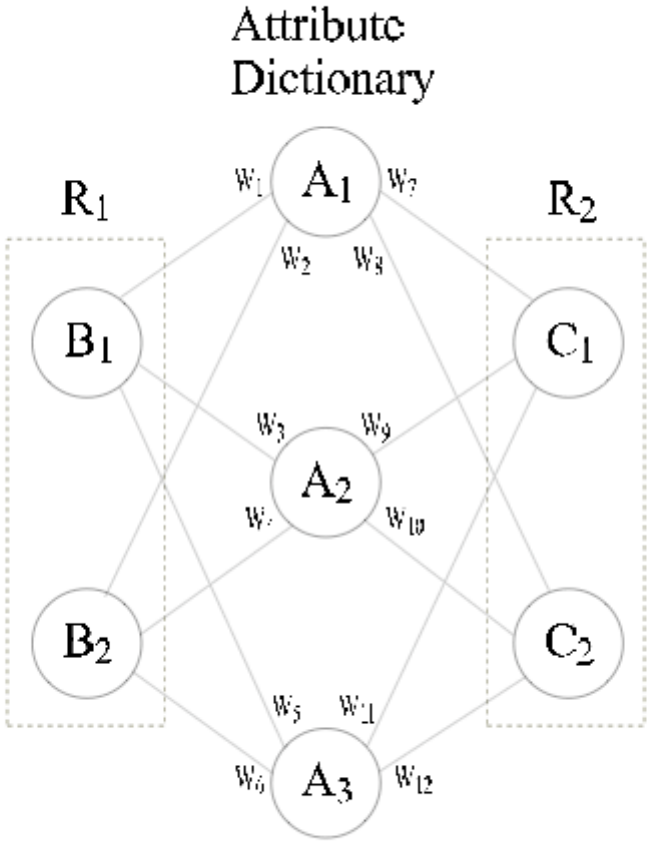
- A set of schema attributes.
- For each attribute there is a set of
  - Possible values
  - Their probability estimates.
- Populated from examples provided by domain experts (using Bayesian learning).
- For each client attribute X (with its set of values) it calculates
  - $M(X,A)$ the probability that X maps to A.
- Details: read the paper.

Optimal Schema Matching

- Assume schemas R_1 and R_2, instances r_1 and r_2, and dictionary D.
- For each attribute X of the schemas and each attribute A of the dictionary calculate the score $M(X,A)$.
- A *threshold* is adopted and scores below it are taken to mean that X could not be mapped to A.
- Results are represented in *weighted tripartite graph*.
  - Because of threshold, the graph need not be *complete*.
- Note: Every two client attributes could be matched through every dictionary attribute.
- To simplify: Limit our search to mappings in which the paths between R_1 and R_2 are free of *intersections*.
  - Two client attributes never map to the same dictionary attribute.

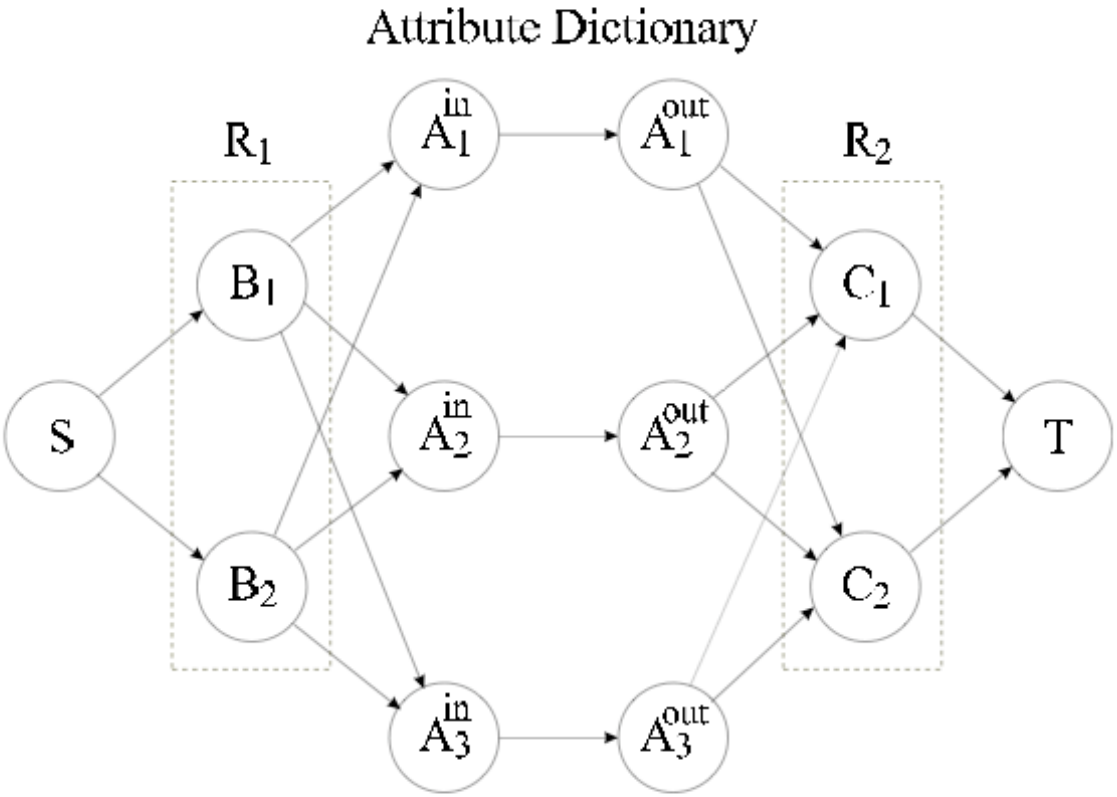
Optimization (*Cont.*)

- Example graph for $R_1=\{B_1, B_2\}$, $R_2=\{C_1, C_2\}$ and $D=\{A_1, A_2, A_3\}$



Optimization (*Cont.*)

- We modify the graph.



Optimization (*Cont.*)

- *Capacity* of each edge: 1
- *Cost* of new edges: 0
- *Cost* of old edges: *negation* of weight
- Use *minimum cost - maximum flow* algorithm
- In the example, maximum flow is 2 (number of attributes to be matched).
- Find edges in the graph that have the minimum cost while supporting the maximum flow of 2.

Optimal Selection of Dictionary Values

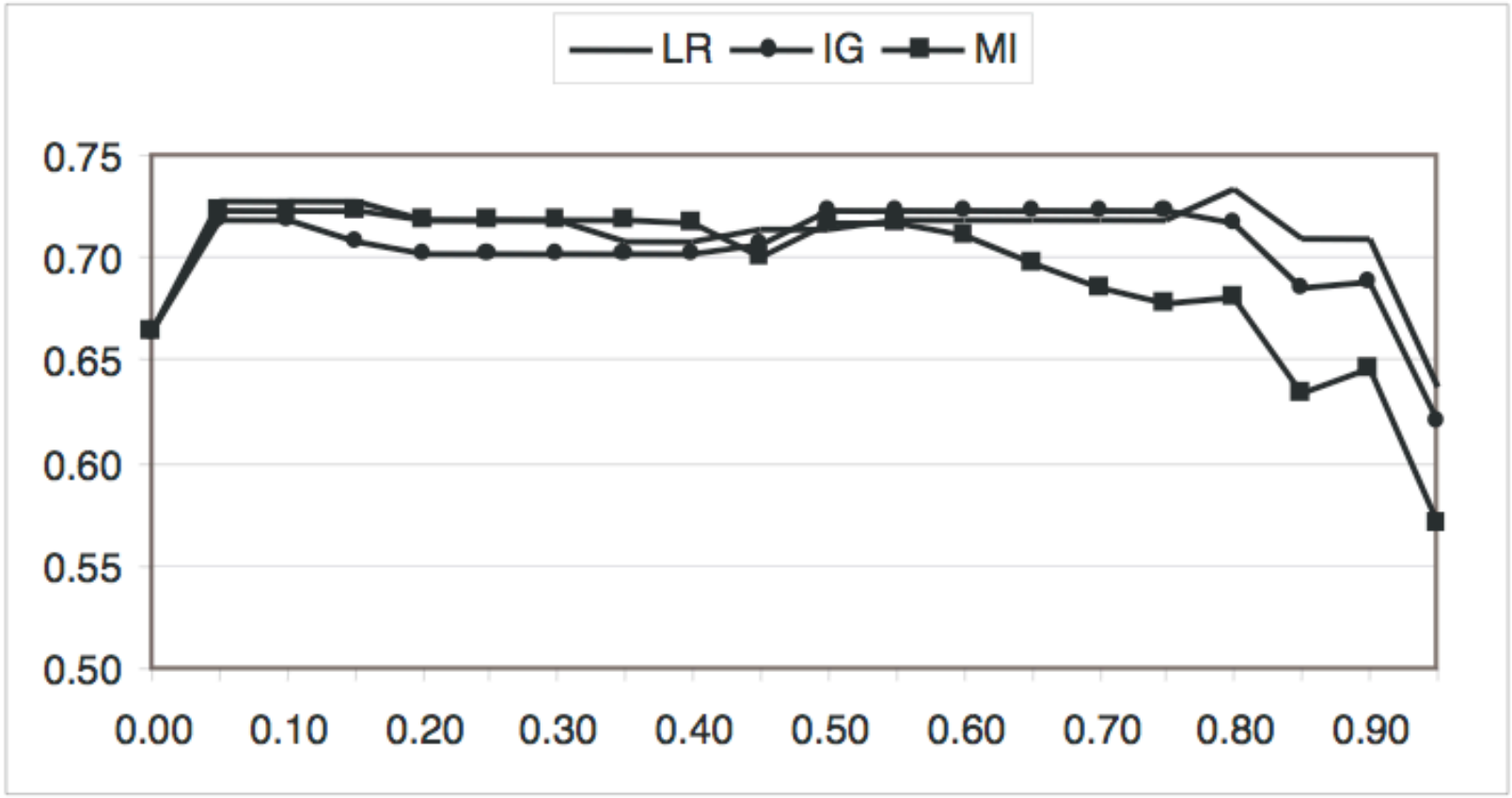
- Techniques for reducing the dictionary representation (store fewer values).
  - Improve storage, computational cost, quality of the results
  - In machine learning terminology: *feature selection*.
- Three methods were compared:
  - Mutual information
  - Information gain
  - Likelihood ratio
- Details: read the paper.

Experimentation

- 22 relations were extracted from 15 web sites of computer retailers.
- Each of the 22 schemas was mapped manually into the attribute dictionary.
- *Stratified cross validation*: Divide the 22 cases into three *folds*:
  - Use two folds for learning and one for testing.
- For the test fold, choose two schemas at a time (for all possible combinations).
- Use Automatch to match the schemas.
- Judge its success in comparison to the manual mappings.
- Measuring success:
  - Each schema mapping is a set of Boolean decisions on all the pairs $<R_1(B_i), R_2(C_j)>$.
  - Decisions fall into one of 4 sets: True positives, true negatives, false positives, false negatives
  - Calculate Soundness and completeness, take their harmonic mean:
    - $F = 2 \cdot (S \cdot C) / (S + C)$

Experimentation (*Cont.*)

- Without dictionary optimization: performance is 66%.
- With 5% through 60% feature reduction: improvement by about 7%.
- With 60% through 80% reduction, MI drops, LR and IG continue well.



II - TupleRank: Ranking Discoveries

- Like all discovery processes, Autoplex content discovery is intrinsically probabilistic:
  - Each discovery is associated with a *predictive value* denoting the level of *assurance* of its appropriateness.
  - Consequently, tuples in a discovered virtual relation have *mixed* assurance levels, with some tuples being more reliable than others.
- Content of discovered virtual databases should be *ranked*.

Tuple Ranking

- Ranking by assurance is useful in several ways.
  1. It is essential for any decision process that might be based on the discovered data.
  2. Users could retrieve from discovered virtual databases on the basis of assurance:
     - Only information that exceeds a certain level of assurance is retrieved.
  3. Ranking of discovered content permits fully-automated data integration.
     - Whereas other systems guarantee “perfect” mappings by using experts, our approach admits mappings of varying quality, relying instead on a grading method.

Calibrating the Predictions

- Yet another source of information is the *cumulative performance* of the Autoplex discovery process.
  - How reliable have the discoveries of Autoplex been?
- Need to combine:
  1. Assurance in each individual discovery (calculated in the process).
  2. Cumulative performance of the system.
- Method for combining: calibrated experts.
  - Assume expert makes a prediction with probability *p*.
  - This expert is *calibrated* if the fraction of true events in the class of events to which the expert assigned a subjective probability of *p*, is equal to *p*.
  - Example:
    - Expert looks at SAT scores and predicts that John will graduate from GMU with probability 0.7.
    - This expert is *calibrated* if 70% of the students to which the expert assigns the probability 0.7 indeed graduate from GMU.
    - If only 60% in this group graduate, then in the future, each time the expert predicts 0.7 it is adjusted to 0.6.

TupleRank

- TupleRank: The Autoplex algorithm for combining individual predictions with the system performance record.
  - Score the training discoveries.
  - Sort and divide into *n* bins of equal size.
  - Average the scores in each bin.
  - Given a new discovery: Substitute its score by the score of its bin.
- Conclusion: Autoplex populates each virtual relation with discovered tuples, in a way that resembles IR search engines.