GLUE and iMAP:

Two ontology mapping tools
Problem: Mapping between ontologies

Goal: To find correspondences between two ontologies

- Uses multiple well-founded similarity measures
- Multiple learning Strategies
- Commonsense Knowledge & Domain Constraints
- Relaxation Labeling to optimize on these constraints
Figure 1: Computer Science Department Ontologies
Meaning of Similarity

• Joint probability distribution to be calculated and used for various measures

• \( P(A,B) \), \( P(A,\bar{B}) \), \( P(\bar{A},B) \), \( P(\bar{A},\bar{B}) \)

• It is difficult to calculate these values, as A and B are usually in isolation

• So, a Classifier of A is trained using instances of A, and then used to classify instances of A in the other ontology.
Different Types of Information

- Name
- Value Format
- Word Frequencies

Best learned by different algorithms
- So GLUE uses multi-strategy learning
- They are combined by a meta-learner

- Finally, Domain Constraints and Heuristics are used to improve the matching accuracy (E.g. location in taxonomy)
- Implemented via relaxation-labeling
 Ontology Matching Problem

An ontology is a model made up of

- Concepts (instantiated to concrete instances)
- Attributes (held by all members of a concept)
- Relations (between concepts)
- Usually organized as a taxonomy

(This is all very similar to the Software Engineering idea of class heirarchies)
Different types of Matching:

- One-to-one: direct matching between elements
- Complex: concatenation and union
- Glue is only worried about the simple (1-1) matchings

Primary Goal:

given two taxonomies and their associated data instances, for each node (i.e., concept) in one taxonomy, find the most similar node in the other taxonomy, for a pre-defined similarity measure.
Similarity Measures:

- Well Defined
- Intuitive (i.e. not just syntactic)
- Broad Variety
Jaccard Similarity:

\[
Jaccard\text{-}sim(A, B) = \frac{P(A \cap B)}{P(A \cup B)}
\]

\[
= \frac{P(A, B)}{P(A, B) + P(A, \overline{B}) + P(\overline{A}, B)} \quad (1)
\]

Most Similar Parent:

\[
MSP(A, B) = \begin{cases} 
  P(A|B) & \text{if } P(B|A) = 1 \\
  0 & \text{otherwise}
\end{cases} \quad (2)
\]

Least Similar Child
Architecture:

- Distribution Estimator - calculate the joint probability using set of base learners and a meta-learner.
- Similarity Estimator - takes user defined algorithm to compute value between each pair of concepts. Represented as a matrix.
- Relaxation Estimator - combines matrix, constraints and heuristics to get a “best match”
Distribution Estimator:

Must assume that our instances are a representative sample

Terms:

- $U_i$ - set of instances given for taxonomy $O_i$
- $N(U_i)$ - size of $U_i$
- $N(U_{i}^{A,B})$ - number of instances in $U_i$, belonging to A and B

So, $P(A,B) = [N(U_{1}^{A,B}) + N(U_{2}^{A,B})]/[N(U_1) + N(U_2)]$

We need to determine $N(U_{1}^{A,B})$ and $N(U_{2}^{A,B})$
For $N(U_2^{A,B})$, we already know the instances $s \mid s \in U_2$ belong to B ($U_2$ is B's taxonomy)

1. Partition $U_1$, into $U_1^A$ and $U_1^\overline{A}$, the set of instances that do and do not belong to $A$, respectively (Figures 3.a-b).

2. Train a learner $L$ for instances of $A$, using $U_1^A$ and $U_1^\overline{A}$ as the sets of positive and negative training examples, respectively.

3. Partition $U_2$, the set of instances of taxonomy $O_2$, into $U_2^B$ and $U_2^\overline{B}$, the set of instances that do and do not belong to $B$, respectively (Figures 3.d-e).

4. Apply learner $L$ to each instance in $U_2^B$ (Figure 3.e). This partitions $U_2^B$ into the two sets $U_2^{A,B}$ and $U_2^{\overline{A},\overline{B}}$ shown in Figure 3.f. Similarly, applying $L$ to $U_2^\overline{B}$ results in the two sets $U_2^{A,\overline{B}}$ and $U_2^{\overline{A},B}$.

5. Repeat Steps 1-4, but with the roles of taxonomies $O_1$ and $O_2$ being reversed, to obtain the sets $U_1^{A,B}$, $U_1^{\overline{A},\overline{B}}$, $U_1^{A,\overline{B}}$, and $U_1^{\overline{A},B}$.

6. Finally, compute $P(A,B)$ using Formula 3. The remaining three joint probabilities are computed in a similar manner, using the sets $U_2^{\overline{A},B}$, $\ldots$, $U_1^{\overline{A},\overline{B}}$ computed in Steps 4-5.

We can do a similar set of operations for all of the joint probabilities.
Multi-Strategy learner:

Many types of information can be exploited

- Word frequency
- Instance names
- Formats
- Characteristics of value distributions
GLUE used two base learners:

A content learner which used naive Bayesian Learning to determine $P(A \mid \text{tokens})$

$$P( A \mid d ) = P( d \mid A ) \frac{P( A )}{P( d )}$$

$P( d \mid A )$ and $P( A )$ are easy to determine

$P( d )$ can be ignored (normalizing constant)

$P( d \mid A )$ is calculated by assuming that tokens appear independently of each other

A name learner which makes its predictions based on the full “path name” from the root of the taxonomy
The meta-learner simply sits on top of the base learners, and assigns weights to their predictions.
Relaxation Labeling:

**Key idea:** the label (or classification) for a particular node is influenced by the features of the node’s neighborhood, in the graph

- Initial labeling is based on intrinsic properties of the nodes
- Iterative local optimization, changes labels based on features in neighborhood.

Take the nodes in $O_2$ as labels and try to assign these to $O_1$'s nodes, based on knowledge we already have
Deriving the formula:

\[ P(X = L | \Delta_K) = \sum_{M_X} P(X = L, M_X | \Delta_K) \]
\[ = \sum_{M_X} P(X = L | M_X, \Delta_K) P(M_X | \Delta_K), \quad (4) \]

Where \( M_X \) is all possible label assignments to nodes other than X in \( O_1 \).
Assuming that label assignments are independent of each other (could be a big assumption), and a given $\Delta k$,

$$P(M_X|\Delta K) = \prod_{(X_i = L_i) \in M_X} P(X_i = L_i|\Delta K). \quad (5)$$

Looking at an instance, $P(X = L|M_X, \Delta K)$

Remember that $M_X$ and $\Delta k$ constitute what we know about the neighborhood of $X$

Suppose that, $P(X = L)$ depends only on the values of $n$ features in the neighborhood, as a function, $f_i(M_X, \Delta K, X, L)$

Then, $P(X = L | M_X, \Delta k) = P(X = L | f_1, \ldots, f_n)$

Using the logistic function, $\sigma(x) = 1/(1 + e^{-x})$, as an estimate of this probability:

$$P(X = L|f_1, \ldots, f_n) \propto \sigma(\alpha_1 \cdot f_1 + \cdots + \alpha_n \cdot f_n), \quad (7)$$

where $\propto$ denotes "proportional to", and the weight $\alpha_k$ indicates the importance of feature $f_k$. 
Figure 4: The sigmoid function
So, \[ P(X = L | \Delta_K) \propto \sum_{M_X} \sigma \left( \sum_{k=1}^{n} \alpha_k f_k(M_X, \Delta_K, X, L) \right) \times \prod_{(X_i = L_i) \in M_X} P(X_i = L_i | \Delta_K) \] (8)

Iterating, using this equation, performs a sort of “hill-climbing” in the problem space

<table>
<thead>
<tr>
<th>Constraint Types</th>
<th></th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Domain-Independent</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Neighborhood</td>
<td>Two nodes match if their children also match.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Two nodes match if their parents match and at least x% of their children also match.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Two nodes match if their parents match and some of their descendants also match.</td>
<td></td>
</tr>
<tr>
<td>Union</td>
<td>If all children of node X match node Y, then X also matches Y.</td>
<td></td>
</tr>
<tr>
<td><strong>Domain-Dependent</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subsumption</td>
<td>If node Y is a descendant of node X, and Y matches PROFESSOR, then it is unlikely that X matches ASSISTANTPROFESSOR.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If node Y is NOT a descendant of node X, and Y matches PROFESSOR, then it is unlikely that X matches FACULTY.</td>
<td></td>
</tr>
<tr>
<td>Frequency</td>
<td>There can be at most one node that matches DEPARTMENTCHAIR.</td>
<td></td>
</tr>
<tr>
<td>Nearby</td>
<td>If a node in the neighborhood of node X matches ASSOCIATEPROFESSOR, then the chance that X matches PROFESSOR is increased.</td>
<td></td>
</tr>
</tbody>
</table>

Each constraint becomes a feature, \( f_i \), with associated weights based upon a judgement of the value of the constraint.
Results:

![Bar chart showing matching accuracy of GLUE.](image)

**Figure 5: Matching accuracy of GLUE.**
• Good accuracy overall

• Name learner alone performed poorly
  ▶ Argument in favor of multi-learning strategy
  ▶ Name learner adds little value normally

• Meta-learner
  ▶ Necessary for multi-learning strategy
  ▶ Only as good as the weighting assignments

• Relaxation Labeler
  ▶ Tendency to “max-out” making it important to know when to stop
  ▶ Tended to get best results by stopping when the mappings themselves stopped changing
Weaknesses:

- Limited to (1-1)
- Insufficient textual data/ no name matches
  - Could add learners to operate more on the structure of the domain
- Local optimization gets stuck in local maximums
  - Simulated annealing?
- Nodes with no match
  - Allow GLUE to predict null matchings
IMAP (Briefly)

Idea: to handle (1-1) and complex matches semi-automatically

Most matchings are currently done by hand

Complexity is the big problem - the number of possible complex matches is essentially unbounded
Modules for iMAP:

- Text searcher - concatenations of text attributes
- Numeric searcher - combine arithmetic attributes via formulae
- Schema mismatch - compare data instances with schema elements (it is quite common for different ontologies to represent differently like this)
- Date searcher - particularly tuned to concatenate date instances

Beam search techniques are used to continually narrow down the search space, with the hope that it will find the best matches
The candidates are then evaluated by a broad array of tools

- Machine Learning
- Statistical Analysis
- Heuristics

The candidates are then re-evaluated with specialized (based upon the type of mapping) criteria and a name similarity algorithm. Domain knowledge is used as soon as possible (even during the original search) to prune the search space

Finally, an explanation module provides feedback, which is useful for fine-tuning

Results aren’t perfect, but it could make a good human aid
Candidate match generation:

- Specialized match generators
- Best choices are collected from *all* generators
- Beam search used to limit the space to the best matches
- Follows law of diminishing returns to halt the search
Similarity estimator:

- Scores specialized matches by
  - Name similarity
  - Naive Bayesian text matcher

Match selector:

- Searches for best similarity scores *given* constraint (also can “clean up” some overly complex matches)
Domain knowledge:

- Domain constraints - rules which are implied by schema or supplied by experts
- Past complex matches - due to similar conversions, a template can ease search
- Overlap data - searches which exploit shared data
- External data - mine data to learn proper distribution of a feature
Explanation module:

Dependency graph is generated during the search process, recording sequence of choices and rational. This is then queried to derive an explanation.
Evaluation and analysis:

- Matching can be provided at different granularities (top-1 top-3 etc.)
- Performance (even on (1-1) concepts) don’t seem sufficient for automatic matching
- Since the authors don’t provide data for above top-3, I would assume that there was no significant gain above this granularity. This bodes poorly even for a module designed to aid human decision making, as it seems very important to have the correct match in the list. For example, a top-10 with close to 100% accuracy would be a very useful tool.