Ontology Matching Using Weighted Graphs

Asankhaya Sharma
National Institute of Technology, India
Warangal asankhaya@gmail.com

Abstract

In this paper, we present a new way towards ontology matching. Using the graph representation for ontologies and schemas we proceed to calculate the weights for each node of the graph using the lexical similarity of the ancestors. With the guiding intuition that, if the parent nodes match then their children are likely to match as well. This simple observation helps on to build a fast and efficient algorithm for matching different graphs (which represent ontology or schema). Since the algorithm is very fast it can be used as a quickly and dirty method to do initial matching of a large dataset and then drill down to the exact match with other algorithms. The algorithm is not dependent on the method used for calculating the lexical similarity so the best lexical analysis can be used to derive the weights. Once the weights are in place we can calculate the matching in just a single traversal of the graphs. No other algorithm that we know of can give such fast response time.

1. Introduction

An ontology is a specification of a conceptualization. It provides a vocabulary that describes a domain of interest and a specification of the meaning of terms used in the vocabulary. With the advent of the semantic web, ontologies have been used to describe what is expressed in computer format. They are viewed as the silver bullet for many applications, such as information integration, peer-to-peer systems, electronic commerce, semantic web services, social networks, and so on [1]. One of the major problem with ontologies can be that anyone can make any number of them, although efforts are being made to standardize them. Even with standards there can be ontologies for similar format derived from different sources.

Ontology matching is one of the hot area of concern to computer scientists all around the world. Several methods are already in place to enable the integration and match making for different ontologies [2]. Ontologies are usually represented as graph like structures (concept hierarchies, classifications, schemas). In this paper we present an algorithm based on calculating the weights in the graph representation of the ontologies. This method as we shall see is a fast and efficient way to match ontologies.

The problem of ontology matching also pops up in a related field of schema integration from different heterogeneous databases. These schemas if represented in a graph form can also be matched using this algorithm. Several of well known algorithms in the semantic integration research have already been found beneficial for the database community [3]. The schema can be conveniently represented as a graph with nodes as the attributes and relations as edges. Here is an example for a table 'Students'.

CREATE TABLE Students (sid CHAR (20),
    name CHAR(20),
    login CHAR(20),
    age INTEGER,
    gap REAL,
    UNIQUE (name, age),
    CONSTRAINT
    StudentsKey PRIMARY KEY(sid))

This schema can be represented as a graph shown below.

![Graph representation of the schema](image)

The attribute nodes can have further children if they refer to a foreign table and so on. This kind of graph can be used to match schemas using the given algorithm.
2. The Algorithm

This algorithm makes use of the simple fact that if node A and A’ match in an ontology represented as a graph then its likely that the nodes B and B’ (the children) will also match.

First of all we match the nodes with some lexical measure. Here we take the variation of the longest common subsequence algorithm to match the nodes according to their lexical similarity; this value is called the lex_sim of a node. The lex_sim of a node can have a value from 0 to 1 depending on the match. The following illustrates the algorithm followed to find out the lexical similarity. The strings X and Y are obtained from the nodes of the graph.

2.1 Calculation of Weights:

**Input:** Strings X and Y with n and m elements, respectively

**Output:** For i=0,...,n-1,j=0,...,m-1, the length L[i,j] of a longest common subsequence of X[0..i] and Y[0..j]

for (i=-1 to n-1) do
  L [I,-1] = 0
for (j=0 to m-1) do
  L [-1, j] = 0
for (i=0 to n-1) do
  for(j=0 to m-1) do
    if (X[i]==Y[j]) then
      L[i, j] = L[i-1, j-1] + 1
    else
      L[i, j] = max{L[i-1, j],L[i, j-1]}
  return array L

L[i,j] gives the length of the longest common subsequence, and from L[i, j] we can find out the lex_sim by dividing the length of X and Y by L[i, j]. So we have,

\[
\text{lex}_\text{sim}(\text{node } A) = \frac{m}{L[i, j]} \\
\text{lex}_\text{sim}(\text{node } A') = \frac{n}{L[i, j]}
\]

where m and n are the lengths of the strings A and A’ respectively.

Now each node has a lex_sim measure with it, we take this lex_sim measure as the node’s approval that its children and grand children will match as well. Thus in a way the lexical similarity of node is propagated to the children. Every node is associated with a weight which is calculated as follows.

\[
\text{weight}(A) = \alpha \cdot \text{lex}_\text{sim}(A) + \beta \cdot \text{lex}_\text{sim}(\text{parent}(A)) + \gamma \cdot \text{lex}_\text{sim}(\text{grandparent}(A))
\]

The above equation calculates the weights for a two generation propagation i.e. up to the grandparent level. Extending it to an ‘n’ ancestor weight we get the following.

\[
\text{weight}(A) = \Sigma n_i \cdot \text{lex}_\text{sim}(\text{ancestor}_i(A))
\]

After the weights have been allotted the ontology graph looks like the one below.

Once we have the weights of each node the matching problem can be tackled without any difficulty. Just making a traversal of the two graphs we match nodes with weights deferring in a small value say ‘d’. This value may depend on the kind of ontology to be matched or the schema to be integrated. This method is similar to clustering the results by taking Manhattan distances. We can take on different values of ‘d’ for experimental data sets before deciding on
the final value. Now we try to use this algorithm on the following ontologies of the departments.

![CS department ontologies](image)

Now moving down the table choosing a value of ‘d’ as 0.4 and traversing the graph in top to bottom left to right we see that this algorithm matches perfectly the faculty and academic staff and then matches Assistant Professor <---> Lecturer, Associate Professor <---> Senior Lecturer, Professor <---> Professor and so on. These matching cannot be derived from the lexical analysis of the similarity measure along. Thus the node ranking gives a way of doing contextual analysis of the ontologies given the lexical similarities. In an exactly similar way we can go about matching the graphs representing schemas.

### 2.2 Algorithm Complexity Analysis

The complexity of the algorithm can be analyzed with respect to two things. One is the lexical matching algorithm and other is the calculation of weights. Since the calculation of weights is not dependent on the type of method chosen to do the lexical matching we can safely assume we have the lexical similarities with us. Then the next step is to construct a table similar to the Table 2. This requires that we traverse the given ontology graphs at least once. Several of the current known algorithms traverse the graphs many times to determine the proper matching [4], [5], and [6]. Once we have the table we need to make one more final traversal of the graph and cluster them as per the chosen ‘d’ values. This cluster represents the matching of the two graphs according to the weights.

### 3. Ontology Matching Results

This algorithm was tested on several ontologies taken from the DAML ontology library and several other ontologies created from dummy data sets. The accuracy of the algorithm was measured using the Longest Common Substring for the lexical matching and then using node ranking. The accuracy is defined with respect to the ability of the algorithm to successfully match more then 50% of the nodes in the given ontologies correctly. Here are the results.
Table 3: Matching results

<table>
<thead>
<tr>
<th>Ontology</th>
<th>Matching</th>
</tr>
</thead>
<tbody>
<tr>
<td>Publications</td>
<td>63%</td>
</tr>
<tr>
<td>Person</td>
<td>71%</td>
</tr>
<tr>
<td>Departments</td>
<td>83%</td>
</tr>
<tr>
<td>Courses</td>
<td>81%</td>
</tr>
</tbody>
</table>

The values in the matching column mean that the algorithm matched more than half of the nodes correctly, that many number of times for the given ontology. Say for courses the given ontologies matched 81% of times correctly (match is defined by matching more than 50% of nodes). The tests conducted involved choosing different values for ‘d’ and changing the number of ancestors up to which to apply the algorithm.

4. Future Work

The tests for various values of ‘d’ and choosing the right ancestor to apply this method are under way. And the results till now are promising. The tests for heterogeneous schemas are also going on. The following are the guidelines for future work.

- Testing the algorithm on a wider range of data sets.
- Testing the algorithm in conjunction with different lexical matching methods to find the one best suited.
- To use this algorithm in existing applications to complement and verify the results, those are already obtained.

5. Conclusion

In this paper we analyzed a new algorithm based on weights which gives a fast and time efficient way to match different ontologies and schemas. As the results suggest, this algorithm can be used as a quick and dirty method to match two ontologies. Since the algorithm fast in comparison to the ones used in current matching applications ([7], [8] and [9]) we can use it to do an initial survey of datasets and identify the graphs that can later be analyzed in depth with other methods.

6. Acknowledgments

We would like to thank DARPA’s Information Exploitation Office which sponsors DAML, which is the source (http://www.daml.org/ontologies) of all the datasets and ontologies used for testing the algorithm.

7. References


