Ontology Driven Information Extraction from Tables Using Connectivity Analysis

Ashwin Bahulkar and Sreedhar Reddy

Tata Consultancy Services Ltd
54 B Hadapsar Industrial Estate
Pune, India
{ashwin.bahulkar,sreedhar.reddy}@tcs.com

Abstract. Table is one of the most common mechanisms used for presenting structured information on the web. A table presents information on a set of related concepts in a domain. A column typically represents a concept or an attribute of a concept that the column header identifies. A row contains corresponding instances and attribute values. However column headers are usually quite noisy and sometimes even missing. While a human reader can figure out the required domain mappings relatively easily by using domain knowledge and surrounding context, discovering them algorithmically poses challenges. In this paper we present an algorithm that exploits the idea that a table only presents information on connected entities of a domain ontology. The algorithm works in two phases. In the first phase it uses local optimization criteria such as lexical matching, instance matching, and so on to find an initial set of mappings. In the second phase it takes these mappings and constructs all possible connected sub graphs of the ontology that can be formed from these mappings. The largest of these sub graphs that has the highest local mapping score is then selected as the underlying domain mapping of the table. We present experimental results demonstrating the effectiveness of the algorithm.

Keywords: Ontology, information extraction, web tables.

1 Introduction

A table is a common means for presenting structured information in documents. Tables vary in complexity, from a simple two-column representation to a large multi-column representation with multi-row headers. The latter are especially common in scientific literature. A scientific domain such as bio-medical or materials science is characterized by a rich underlying ontology. In these domains a table may present information spanning multiple entities in the ontology. For example, in the materials domain, a table may present information on materials, their compositions, properties

R. Meersman et al. (Eds.): OTM 2013, LNCS 8185, pp. 642–658, 2013.

Table 1. Table with unlabeled columns

Physical Property		
Density	0.84	[g/cm³]
Specific Volume	1.19	[cm³/g]

Table 2. Table with misleading column headers

Quantity	**Value**	**Unit**
Thermal Expansion	10-13	e-6/K
Thermal Conductivity	50-50	W/m.K
Specific heat	460-540	J/kg.K

Table 3. Table from Transport domain

No	**Name**	**Type**	**Zone**	**From**	**Dep**	**To**	**Arr**	**Duration**	**Halts**
19019	Mumbai Exp	Exp	WR	BDTS	00:05	NZM	05:25	29h 20 min	74
02455	Swaraj Express	SF	NR	BDTS	06:25	NDL	07:40	25h 15m	7
19023	Delhi Exp	Exp	WR	BDTS	07:25	NDL	12:45	29h 20min	67
12471	Rajdhani Exp	SF	WR	BDTS	07:55	NZM	04:30	20h 35min	15

of the materials and the conditions under which those properties are exhibited. This maps to several entities, their attributes and relationships in the materials ontology. These mappings are typically indicated by column labels. However, there is usually a large variation in the kinds of labels used. Short hand notations, truncated strings, etc are quite common. Sometimes labels may even be completely missing. These variations exist not only because there is no standard terminology for labels, but also to satisfy requirements of presentation such as layout constraints, etc. Despite this noise, a human reader can usually figure out the mappings easily, as he/she does not go by the labels alone but uses his/her domain knowledge and the knowledge of the context in which the table is presented. Hence an automated extraction algorithm that relies solely on labels and other local cues will not do well in the presence of such noise, especially when complex tables are involved. A table usually presents information on related entities. Many ambiguous situations can be easily resolved if we make the hypothesis that all information in a table is related. We call this "connectivity hypothesis". For example, in Table 2 we have a column named "Quantity". This does not map to any term in the materials ontology, a fragment of which is shown in Fig 1. However a table level connectivity analysis tells us that this most probably maps to "Property" as the other two columns are mapped to "Property Value" and "Unit". In Table 1, column 1 is labeled "Physical Property" while columns 2 and 3 are unlabeled. With local analysis (instance analysis) we can figure out that column 3 has units and thus label column 3 as "Unit". At this stage table connectivity analysis with respect to the ontology tells us that the unlabeled column can only map to "Property Value".

Table 3 shows a table from transportation domain. Column 2 is labeled "Name" which could potentially refer to name of a place, train or some other means of transportation. Columns after it are labeled "arr time", "dep time", "from" and "to". Since these are only related to transportation mode, global analysis rules out place as a possible match for column 2. Looking further at the column labeled "zone" global analysis can infer that column 2 can only map to Train as zone is an attribute of Train.

As illustrated in the previous examples, connectivity hypothesis and global analysis not only help us figure out correct mappings for columns, but also figure out the relationships between the columns.

We present an algorithm that exploits the connectivity hypothesis to figure out correct mappings. The algorithm proceeds in two steps:

Step 1
In the first step, it uses local optimization criteria such as lexical matching, instance matching, and so on to find an initial set of mappings. Columns are mapped to classes and attributes in the ontology based on local cues such as the header and contents of a column. Lexical matching consists of matching the column header against the names of all classes and attributes in the ontology and selecting those that match above a threshold. Instance matching consists of matching cells against known instances in the dictionary. Cell contents are also matched against regular expressions and data types defined in the ontology. A weighted score is computed from these individual scores.

Step 2
In the second step, we resolve ambiguous mappings and find mappings for unmapped columns by doing global connectivity analysis. This is done by trying to find the largest connected sub graph of the ontology that covers the mappings discovered in the local matching step. There may be several candidate graphs of same size. We break the tie by taking the graph that has the highest aggregate local matching score.

Our experimental results show that this global approach does significantly better than approaches that rely purely on local cues. The improvement is more pronounced in the case of complex tables. We obtained an accuracy value close to 80% with the global approach as against an accuracy of around 60% with local approaches.

1.1 Application Context

The approach presented here is developed as part of an information management system that extracts and integrates information of a scientific domain from a number of heterogeneous sources. Queries in this domain can be complex spanning multiple entities with multiple conditions. For example, Table 5 shows information about 5 different concepts – an alloy, its composition, its properties, property values and the conditions under which those values are obtained. A typical query could ask for the name of an alloy with a particular composition, exhibiting a certain property at a

certain temperature. Often information to satisfy such queries does not come from a single source. We might have to integrate information extracted from multiple websites.

Many of these websites are form based which return documents containing tables. The approach presented here is used for discovering mappings between such tables and ontology elements. These mappings, once discovered, can be reused for all future extractions from the site as pages are typically generated from the same underlying template.

2 Related Work

Limaye *et al.* [1] discuss a machine learning approach that uses a probabilistic graphical model for simultaneously choosing entities for cells, types for columns and relations for column pairs. As with any machine learning approach, the success of this depends on the quality of the seed catalog used for training. We propose a heuristic approach that exploits the connectivity hypothesis which seems to work quite well for complex tables.

Cafarella *et al.* [2] discuss an approach for querying the web for relevant web tables. Their approach collects corpus-wide statistics on co-occurrence of schema attributes, and uses these statistics to improve search relevance, synonym finding and join processing. The approach is meant primarily for returning right set of web tables to the user and not so much for automatically extracting information from these tables.

Embley *et al.* [3] present a set of heuristics for filtering relevant tables in HTML documents. We use a set of heuristics that are similar for filtering out irrelevant tables. The paper also presents an approach for extracting information from the tables. But they assume that each table presents information about just one fundamental concept and its attributes. Information on related concepts is expected to be found on separate linked pages. This is a limitation for processing complex tables such as the ones discussed in this paper where information on several related concepts is presented together.

Wang et al. [4] discuss an ontology based approach for extracting information from complex tables. They also present a grammar for complex table layouts and rules for transforming them into a standard form that is amenable for querying. They use ontology to identify concepts and concept instances represented by table cells. Concepts are recognized by their names and possible synonyms and instances are recognized by names or patterns. Where ambiguity arises, they use ontological closeness, i.e. how closely related the concepts of two adjacent cells are in the ontology hierarchy, to resolve the ambiguity. This is different from our approach where we exploit ontology to resolve ambiguities globally (i.e. not just adjacent cells) using the connectivity hypothesis.

A.Pivk et al.[8] discuss an approach to convert web tables into F-logic frames. While they exploit the information recorded in the WordNet ontology such as hyponym, hypernym, etc, the approach does not work with domain specific ontologies and so does not exploit the rich relational information present in them.

Web wrappers have traditionally been used to extract information from semi structured data sources, particularly web pages [5]. Wrappers evolved from being manually written to now being discovered through machine learning approaches. DIADEM [6] discusses an approach to extract data from semi structured web pages using an ontology centric approach. However this does not cover extraction from web tables.

We have not come across any work that uses connectivity analysis at a global level to figure out mappings for information extraction the way we do.

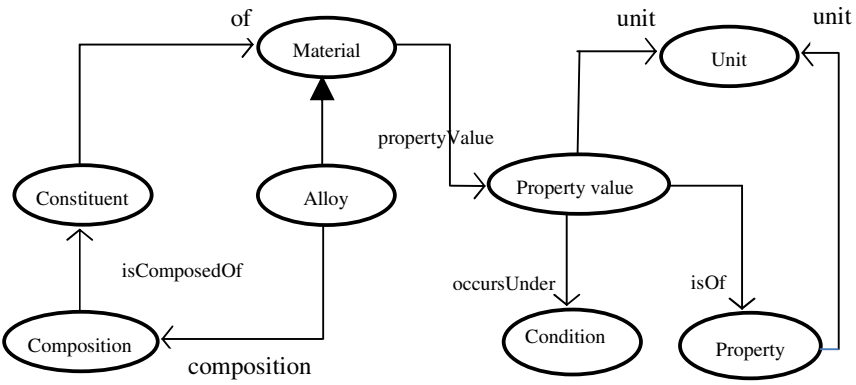

Fig. 1. The materials ontology

3 Ontology Enrichment

An ontology provides a conceptual model of a domain and defines the terminology for concepts, properties and relationships that exist in the domain. Fig 1 shows a fragment of the materials ontology we refer to in our examples. We enrich the standard ontology with the following annotations:

1. Synonyms of classes and attributes.
2. Regular expressions: to specify patterns for attribute values and instance names. We have extended the regular expressions language a little to allow expressions to refer to Ontology terms. For example, instances of the Property Value class might contain property values (numbers) followed by units, e.g. a weight value could be written as “5 kg”. In this case, a regular expression could be written as “[0-9]+ Inst(Unit)”. Here “Inst” specifies that the string should match with an instance of the class Unit. The string “5 kg” would match the regular expression if “kg” is known to be an instance of the class Unit.

4 Mappings Model

Fig 2 shows the model used for capturing mappings between a table and the domain ontology. A table may be embedded in a context that identifies an ontology object on which the table provides additional information. A column may be mapped to a class, attribute or object. A cell may be mapped to an object or an attribute value. When a column is mapped to a class, corresponding cells are mapped to objects of that class. When a column is mapped to an attribute, corresponding cells map to values of the attribute for objects contained in a related column. When no such related column exists, the attribute is assumed to be of the context object the table maps to. When a column maps to an object, corresponding cells map to objects that are related to the column object; the relationship is identified by the corresponding association object. The association object identifies the class of the column object, class of the cell objects and the association between the two classes. Two columns may be related to each other either by an association relationship or an attribute relationship; in the former case the two columns must map to the corresponding end classes of the identified association; in the latter case one column must map to a class and the other to an attribute of that class. A column may also be related to the context object of the table. Again this may be an association or attribute relationship with the same semantics as discussed above. The model in Fig 2 is intended for the case where columns are mapped to ontology elements. A similar model exists for the case where rows are mapped to ontology elements. We don't discuss this case in the paper as the treatment is similar.

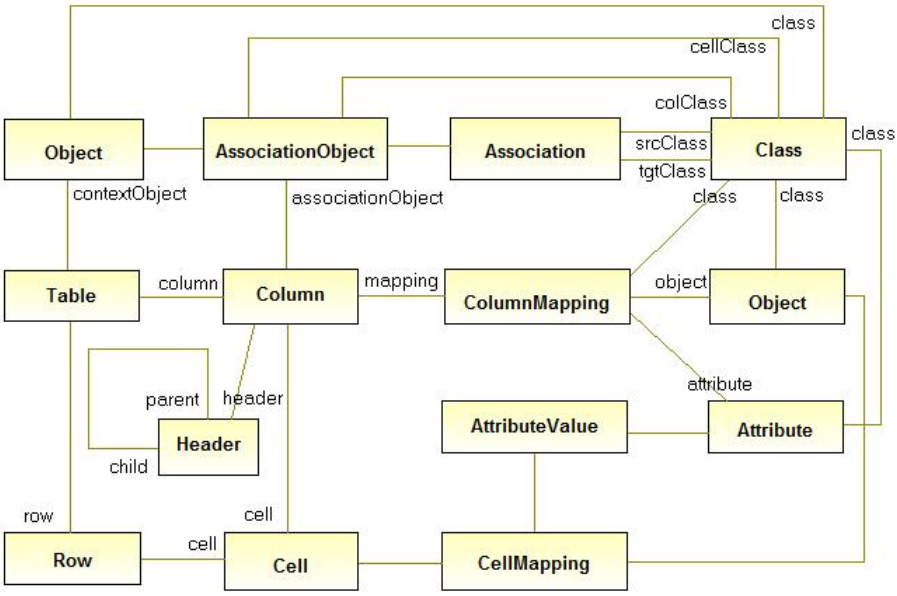

Fig. 2. Mappings Model

The objective of the mapping discovery algorithm is to discover an instance of this mappings model that is as close to the intended mapping as possible.

Before running the algorithm we preprocess the table to bring it into a normalized form. For instance we use the columnspan attribute to detect multi-row headers. A cell with a columnspan greater than 1 indicates that it has sub headers below it. We split such a cell into as many copies as the columnspan. Thus we only need to look at the cells in a single column to retrieve the header hierarchy of that column. There are a few other preprocessing steps such as removing visual attribute tags, etc. which we are not discussing further in the paper.

5 Algorithm for Discovering Mappings

A table generally presents information on related entities. We present an algorithm that exploits this notion of connectedness of information to figure out mappings. The algorithm proceeds in two phases:

Phase 1
In the first phase it uses local optimization criteria such as lexical matching, instance matching, and so on to find an initial set of mappings. Columns are mapped to classes and attributes in the ontology based on a score computed from local cues such as header matching and content matching consisting of instance matching, type matching, regular expression matching, etc.

Phase 2
In the second phase, we resolve ambiguous mappings and find mappings for unmapped columns by doing global connectivity analysis. This is done by trying to find the largest connected sub graph of the ontology that covers the mappings discovered in the local matching step. There may be several candidate graphs of same size. We break the tie by taking the graph that has the highest aggregate local matching score.

5.1 Phase 1 - Finding Local Cues

The following procedure computes a weighted sum of scores from various local cues. We try to find matching classes and attributes for each column. Class matching includes lexical matching as well as instance matching. Only classes whose lexical matching score is above a given threshold are considered as likely candidates. Attribute matching includes lexical matching, type matching and regex pattern matching. Only attributes whose lexical matching is above a given threshold are considered as likely candidates. For lexical matching we use a standard edit-distance based algorithm [7]. We also match column header against instance objects recorded in domain dictionary. When a column header maps to an instance object, we expect the cells to map to instance objects of a related class. This is done by taking the class of the header object and looking for classes related to it in the ontology. The classes whose

instance matching and regex matching scores are above a threshold are considered as probable candidates for the cells.

The procedure finds all probable matches that satisfy threshold constraints. These are then pruned in the next phase.

The algorithm is presented in more detail below. In the algorithm we treat the model given in Fig 2 as a data structure, where associations and attributes are treated as members of the corresponding structure element.

```
procedure ComputeLocalMatchingScore
Input: Table tbl, Ontology ont
Output: A set of ColumnMapping elements
Begin
   For each column col in tbl.column do
      // Step 1: find potential matching classes
      For each class cls in ont do
          lms = LexMatch(col.header, cls.name)
          if (lms > lexicalMatchingThreshold) then
              // Get the percentage of column cells that
              // match instances of the class in the do
              // main dictionary
              ims = InstMatch(col, cls)
              // Get the percentage of column cells that
              // match the class name patterns, if
              // any specified for the class.
              rms = RegExMatch(col, cls)
              ws = lexMatchWeight * lms
                      + instMatchWeight * ims
                      + regexMatchWeight * rms;
              cm = new ColumnMapping
              cm.mappingScore = ws
              cm.class = cls
              col.mappimg = cm
          end if
      end for
      // Step 2: find potential matching attributes
      For each attribute attr of each class in ont
      do
          lms = LexMatch(col.header, attr.name)
          if (lms > lexicalMatchingThreshold) then
              rms = RegExMatch(col, attr)
             ws = LexMatchWeight * lms +
                   RegexMatchWeight * rms
              cm = new ColumnMapping
              cm.mappingScore = ws
              cm.attribute = attr
```

```
            col.mappimg = cm
        end if
    end for

    // Step 3: Next try to match column header with
    // objects (of any class) recorded in the domain
    Object objList[] = FindMatchingObject(col)
    For each obj in objList do
        cm = new ColumnMapping
        cm.object = obj
        col.mapping = cm

        // When a column header maps to an object, its
        // cells are expected to match with objects of
        // a related class. We try to find such a
        // matching related class. Again there could be
        // more than one probable match.
        For each relCls in
               obj.class.association.tgtClass
        do
           // Consider classes whose combined instance
           // and regex matching scores are above a
           // threshold. Instance and regex matching
           // steps are as discussed previously for
           // class matching.
           instScore = getInstanceMatchScore(col,
                                          relCls);
           if (instScore > InstMatchThreshold) then
               AssociationObject ascObj =
                           new AssociationObject()
               ascObj.association = assoc with relCls;
               ascObj.columnClass = obj.class;
               ascObj.cellClass = relCls;
               col.associationObject = ascObj;
           end if
        end for
    end for
End

Procedure for lexical matching in a column:
procedure LexMatch
Input:  Header hdr, Class cls
Output:  matchScore : float.
Begin
    // Where we have multi-level headers, we start by
```

```
    // looking for a match for the most concrete class at
    // the leaf level and go up the hierarchy until we
    // find a match
    Header leafHdr = getLeafHeader(hdr);
    List clsNameList = getAllSynonyms(cls.name)
    while leafHdr is not NULL do
       // Get max matching score from the given list
       score = getEditDistance(leafHdr.label,clsNameList)
       if (score > lexicalMatchThreshold) then
          return score;
       end if;
       leafHdr = leafHdr->parent;
    End while
    return 0;
End
```

Illustration for Local Analysis

We illustrate how the algorithm works for the Table . 3. The table belongs to the Transport Ontology, which contains the classes Train, Bus, Flight journeys, each having attributes like arrival time, departure time, origin and destination place etc. The table gives information about different trains. Lexical matching recognizes the column headers "No","Name","Zone", "From", "To", "Arr", "Dep", and it maps these columns to ontology attributes. Each mapping has a score which is based on regular expressions and instance matching for the cells in the column, for example the attribute "Arrival time" is associated with a regular expression, [0-2][0-3]:[0-5][0-9] which specifies clock time, and all the cells under the "Arr" column confirm to this regular expression. However, ambiguity exists because all of these attributes do not belong only to the Trains class in the Transportation ontology, they could belong to the Bus as well as Flight class. Only the "Zone" attribute is unique to the Train class. This ambiguity is resolved in the further stages of the algorithm.

In Table. 1, column labeled "Physical Property" is mapped to the Property Class, in the Material Science ontology, given in Fig 1. The 2nd and 3rd columns do not have a header, so they cannot be mapped to any ontology class or attribute in the local analysis phase. Mappings for these columns can be discovered only in the global analysis phase.

5.2 Phase 2: Analyzing Global Cues

This phase has two steps. In the first step we try to find mappings for columns that were not mapped in phase 1. This is done by trying out ontology elements that are in the vicinity of elements that are already mapped. This follows the intuition that a table presents information only on related ontology elements. Here again we try to find all probable matches for an unmapped column subject to threshold constraints.

In the second step, we try to discover the largest connected graph among the mappings discovered in the previous steps. This is done by taking each combination of possible column mappings and constructing connected graphs among them. A connected graph is constructed by taking the set of mapped classes and trying out all possible edges (corresponding to associations in the ontology) among them and taking the set of edges that results in the largest graph. After enumerating all such connected graphs among all possible combinations, we take the largest such connected graph as the final mapping. The largest connected graph may not be unique as there can be several graphs of the same size. We break the tie by taking the graph that has the highest aggregate local matching score.

Vicinity Analysis to Map Unmapped Columns

```
Procedure mapUnmappedColumns()
Input:
    Partially mapped table tbl, Ontology ont
Output:
    Table tbl with more refined mapping
Begin
    List<Column> colList = getUnmappedColumns(tbl)
    // Get the list of classes to which columns have
    // already been mapped.
    List<Class> clsList = getMappedClasses(tbl)

    // First check if the column matches with an
    // attribute of an already mapped class; record all
    // such matches as potential mappings.
    For each col in colList do
        For each cls in clsList do
            // First check if the column matches with an
            // attribute of an already mapped class;
            // record all such matches as potential
            // mappings.
            For each attr in cls.attribute do
               // Attribute matching score is computed as
               // discussed in function
               // ComputeLocalMatchingScore, except that
               // here there is no lexical matching
               score = GetAttributeMatchingScore(col,
                                                 attr)
               if (score > columnMatchThreshold) then
                  cm = new ColumnMapping
                  cm.mappingScore = score
                  cm.attribute = attr
```

```
                col.mappimg = cm
              end if
            end For

            // Next check if the column matches with an
            // associated class of an already mapped
            // class; record all such matches as
            // potential mappings.
            For each aCls in cls.association.class do
              // skip if an already mapped class
              if aCls is in clsList then
                  skip; // go to the next iteration
              end if
              // Class matching score is computed as
              // discussed in function
              // ComputeLocalMatchingScore, except that
              // here there is no lexical matching
              score = GetClassMatchingScore(col, cls)
              if (score > columnMatchThreshold) then
                  cm = new ColumnMapping
                  cm.mappingScore = score
                  cm.class = aCls
                  col.mapping = cm
                  skip;
              end if
              // Next check if the column matches with an
              // attribute of the associated class of an
              // already mapped class; record all such
              // matches as potential mappings.
              For each attr in aCls.attribute do
                score = GetAttributeMatchingScore(col,
                                                  attr)
                if (score > columnMatchThreshold) then
                   cm = new ColumnMapping
                   cm.mappingScore = score
                   cm.attribute = attr
                   col.mappimg = cm
                end if
              end For
            End For
        End For
    End For
End
```

Illustration for Vicinity Analysis

The Table 1, has 2 unmapped columns, and the 1st column has been mapped to the Property class. The Vicinity analysis phase then finds mappings for the 2nd and 3rd column in the vicinity of the Property class. Thus the classes Property Value and Unit are potential mappings for the 2nd and 3rd columns. For the 2nd column, the class Property value has a high local matching score, because most cells conform to the regular expressions specified for the Property Value class. The 3rd column maps to the Unit class because it contains several known instances of the Unit class.

Connectivity Analysis to Discover the Largest Connected Graph

The previous stages of the algorithm have yielded mappings for each column of the table along with the confidence levels for each mapping. A column may be mapped to multiple classes and attributes. These classes form the nodes of the potential mapping graph.

Our objective is to obtain the largest of these potential mapping graphs with the best local matching score. I.e. we want to discover a graph that maps the largest number of columns to ontology elements that form a connected graph. There can potentially be more than one connected graph of the same size. Among them we choose a graph whose nodes have the highest aggregate local matching scores.

```
Procedure FindMappingGraph
Input: Table tbl
Output: mapping graph
Begin
   Struct ColumnMapping {Column col, Class cls}
   // Generate all possible combinations of column-class
   // mappings. In the data structure below each
   // combination is a list of column-class mappings;
   // each such combination will have exactly one element
   // for each column.
   Struct ColumnMappingCombination List<ColumnMapping>
   List<ColumnMappingCombination> listOfColMapCmb =
           generateMappingCombinations(tbl)
   List<Graph> potentialGraphsList
   For each colMapCmb in listOfColMapCmb do
      Graph connectedGraph =
         getLargestConnectedGraph(colMapCmb)
      potentialGraphsList.add(connectedGraph)
   End For


   // By largest connected graph we mean a connected
   // graph with the largest number of nodes.
```

```
    List<Graph> largestGraphsList =
       findLargestGraphs(potentialGraphsList)
    float largestScore = 0;
    Graph selectedGraph;
    For each grph in largestGraphList do
      float curScore = getAggregateLocalScore(grph)
      if (curScore > largestScore) then
        selectedGraph = grph
      end if
    End For
    return selectedGraph
End
```

Illustration for Global Analysis

The global approach resolves the ambiguity introduced in Phase 1. Since there are multiple number of mappings for many columns, several potential graphs can be generated. Out of all of these, the graph which maps the columns labeled "Arr", "Dep", "From","To" to attributes of the Train class has the largest size because the column labeled "Zone" maps to only the attribute Train.zone, whereas the graphs which map the columns to attributes of the Bus or Flight class have smaller graph sizes, they map lesser number of columns. In addition to this, the column labeled "Name" contains the names of a few trains, which increases it's local matching score.

6 Experimental Results

We summarize the results of our experiments in this section. We have experimented with tables of varying complexity drawn from a number of pages of different websites corresponding to three different domains, namely materials science (mat-web.com, copper.org, matbase.com, keytometals.com, polymer.nims.go.jp), transport (Eurostar.com, indiarailinfo.com, Neetabus.in, seat61.com) and meteorology (imd.gov.in, climatemps.com, accuweather.com). We classified the tables into three categories: simple, moderately complex and complex. A simple table represented a single class and its attributes. A moderately complex table had two to three classes, their attributes and relationships. A complex table had four or more classes, their attributes and relationships. To give an example of a complex table, Table 5 has 6 classes and 6 associations with nested column headers. We have collected results for three different configurations of the algorithm:

- **Local-only mode:** only local cues, namely lexical matching, regular expressions and instance matching are used. I.e. only phase 1 of the algorithm is run.
- **Local+column-neighborhood analysis.** Here in addition to the above we also try to find mappings for unmapped columns by searching in the immediate vicinity of classes to which left and right columns are mapped. This is similar to the approach used in [4].
- **Local+global connectivity analysis.** This is the full-fledged algorithm where we try out various connected sub graphs to find the best matching subgraph.

Table 4 summarizes the results. Overall, across all tables, local-only approach gave us an accuracy score (f1-score) of 58.6%, local+column-neighborhood analysis an accuracy of 66.1% and local+global analysis an accuracy of 81.8%. For simple tables these figures were 80, 85.7 and 87.6 respectively, whereas for complex tables corresponding figures were 45.3, 48.9 and 72.2. The figures show that while the overall accuracy is much higher for simpler tables, the algorithm variants do not make that much of a difference. This is to be expected as connectivity analysis does not play a big role in the single class case, as mappings are limited to a single class. In the case of complex tables the global connectivity algorithm makes a big difference as can be seen from the score jumping from 45.3 to 72.2. This is because complex tables present a lot of ambiguous scenarios stemming from inaccurate and missing labels. Global connectivity analysis helps pin down such ambiguous cases by placing them in the connected context of the rest of the graph. The results also show that in the case of complex tables, adding column-neighborhood analysis does not improve the situation as substantially as the global connectivity analysis. To see why, let's look at columns "Dep" and "Arr" in the example of Table 3. By looking at their left and right columns we won't be able to resolve them to Departure and Arrival attributes of class Train. They could be the attributes of any other mode of transport. It is only through global connectivity analysis that we will be able to identify them as the attributes of Train which in turn we were able to identify as the only mode of transport in the table because 'zone' is an attribute only of the class Train.

In our experiments we tried out the following weightages for different local matching criteria:

Case 1: lexMatchWeight = 0.5; instMatchWeight = 0.25; regexMatchWeight = 0.25

Case 2: lexMatchWeight = 0.75; instMatchWeight = 0.125; regexMatchWeight = 0.125

Case 3: lexMatchWeight = 0.25; instMatchWeight = 0.375; regexMatchWeight = 0.375

Case 1 had an accuracy of 81.88%, case 2 had 80% and case 3 had 78%. These weightage differences did not have any substantial impact on the results (within an accuracy variation of ~4%). This reaffirms that graph connectivity is the dominant factor by far in resolving the ambiguous cases.

We have also investigated how the accuracy varies with the parameter *lexicalMatchingThreshold*. This value determines how strictly lexical matching is carried out. Lower values will result in more potential matches per column and hence increased ambiguity. In our experiments we observed for a *lexicalmatchingThreshold* value of 0.3 we got on an overall accuracy of 74%, whereas with a *lexicalmatchingThreshold* of 0.8 we got an overall accuracy of 81%. This shows the global connectivity algorithm is quite robust for large variations in ambiguity.

Results given in table 4 were obtained with the configuration <*legicalMatchingThreshold*=0.8, *lexMatchWeight*=0.5, *instMatchWeight*=0.25, *regexMatchWeight*=0.25>.

Table 4. F1 scores for various kinds of tables

	No. of tables	Local + global connectivity analysis	Local + Column-neighborhood analysis	Local-only
All tables	28	81.88	66.1	58.5
Simple tables	6	87.6	85.7	80
Moderately comlex Tables	13	82	72	58.6
Complex tables	9	72.2	48.9	45.3

Running Time. The complexity of the algorithm can increase significantly as we go from the local-only approach to global connectivity analysis. For an ontology of size N nodes and a table of M columns, the worst case complexity can be N**M. However, in practice a column does not have more than 3-4 ambiguous matches, so the average complexity is within acceptable limits. For example, in the case of the table in Table 5 from the materials domain, the running time for global connectivity analysis was 0.326 seconds as compared to 0.269 seconds for the local-only mode. This table has 14 columns and the corresponding ontology has 15 classes, 20 attributes and 21 associations.

Table 5. Complex Table

Copper and Copper Alloy		Condition	Composition, %						Test Temperature, K	Plastic Properties			
No	Name		Pb	Fe	Sn	Zn	Ni	P		Tensile Strength, psi	Yield Strength, psi	Elongation, % in 4D	Reduction of Area, %
102	Oxygen free	Cold Drawn 60%	4 ppm	4 ppm	1 ppm		4 ppm	1 ppm	295	48400	46800	17	77
									195	52900	49800	20	74
									76	66400	54400	29	78
									20	74500	58500	42	76

7 Conclusions and Future Work

We have presented an ontology based algorithm for information extraction from tables. The algorithm discovers mappings between table columns and ontology elements. It exploits the notion of connectedness of information in a table to resolve ambiguous mappings. The algorithm uses several strategies for local matching such as lexical matching, instance matching, regular expression pattern matching and so on. Despite the sophistication of local matching strategies complex tables usually throw

up many ambiguous cases. Global connectivity analysis makes a substantial difference in resolving such cases. Our experimental results bear this out.

Going forward we would like to benchmark the effectiveness of our approach against machine learning based approaches such as the one discussed in [1]. We also want to test the algorithm on tables found in scientific publications. We want to integrate the table extraction algorithm with text based information extraction algorithms. We believe this integrated approach will increase the accuracy further as we can obtain additional context from the surrounding text.

References

1. Limaye, G., Sarawagi, S., Chakrabarti, S.: Annotating and Searching Web Tables Using Entities, Types and Relationships. Proceedings of the Very Large Data Bases Endowment 3(1) (2010)
2. Cafarella, M.J., Halevy, A., Wang, Z.D., Wu, E., Zhang, Y.: WebTables: Exploring the Power of Tables on the Web. In: Very Large Data Bases, Auckland, New Zealand (2008)
3. Embley, D.W., Tao, C., Liddle, S.W.: Automating the Extraction of Data from HTML Tables with Unknown Structure. Data & Knowledge Engineering - Special Issue 54(1) (July 2005)
4. Wang, H.L., Wu, S.H., Wang, K.K., Sung, C.L., Hsu, W.L., Shih, W.K.: Semantic Search on Internet Tabular Information Extraction for Answering Queries. In: Proceedings of the ACM CIKM International Conference on Information and Knowledge Management (2000)
5. Chang, C.-H., Kayed, M., Girgis, M.R., Shaalan, K.: Survey of Web Information Extraction Systems. IEEE Transactions on Knowledge and Data Engineering (2004)
6. Furche, T., Gottlob, G., et al.: DIADEM: Domain-centric, Intelligent, Automated Data Extraction Methodology. In: World Wide Web Conference – European Projects Track (2012)
7. Levenshtein distance. In: Black, P.E. (ed.) Dictionary of Algorithms and Data Structures, August 14, U.S. National Institute of Standards and Technology, Algorithms and Theory of Computation Handbook. CRC Press LLC (2008) (accessed October 31, 2011)
8. Pivk, A., Cimiano, P., Sure, Y.: From Tables to Frames. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 166–181. Springer, Heidelberg (2004)