# A Semantic Message Passing Approach for Generating Linked Data from Tables

Varish Mulwad
Program: Ph.D., Computer Science
Ebiquity Research Lab
Advisor: Dr. Tim Finin
varish1@cs.umbc.edu

## ABSTRACT

Large amounts of information is stored in tables, spreadsheets, CSV files and databases for a number of domains, including the Web, healthcare, e-science and public policy. The tables' structure facilitates human understanding, yet this very structure makes it difficult for machine understanding. We describe work on making the intended meaning of tabular data explicit by representing it as RDF linked data, potentially making large amounts of scientific and medical data in important application domains understandable by machines, improving search, interoperability and integration. Our domain-independent framework uses background knowledge from the LOD cloud to jointly infer the semantics of column headers, table cell values (e.g., strings and numbers) and relations between columns and represent the inferred meaning in RDF. A table's meaning is thus captured by mapping column headers to classes in an appropriate ontology, linking table cell values to literal constants, implied measurements, or LOD entities (existing or new) and discovering or identifying relations between columns. At the core of our framework is a probabilistic graphical model that exploits existing LOD knowledge to improve a message passing scheme during the joint inference process. We have evaluated or framework on tables from the Web and Wikipedia with promising results.

## Keywords

Tables, Semantic Web, Linked Data, Graphical models

## 1. INTRODUCTION

Tables form an integral part of documents, technical reports, Web pages, and papers, often recording and encoding important information which could not represented be in other forms. When not embedded in documents, table-like structures such as spreadsheets, CSV files and databases are used to capture and represent information. Tables are known to be ubiquitous in a number of domains including the Web, healthcare, e-science and public policy. A Google study [4] showed that the Web alone had more than 150 million high quality relational tables, conveying important and useful information. Various nations around the world, including the United States, share data that can be useful in informing public policy in structured formats such as CSV files. As of September 2012, the US government's data sharing website has nearly 400,000 such datasets.

Both integrating and searching over this information benefits from a better understanding of its intended meaning. Analyzing tables presents several unique challenges. The very structure of tables which adds value and makes it easier for human understanding also makes it harder for machine understanding. Take the example of Web search engines. They do an excellent job when searching over narrative text on the Web, but perform poorly when searching for information embedded in tables in HTML documents.

In the domain of evidence-based medicine [20], medical researchers attempt to judge the efficacy or drug dosages or treatment by performing a meta-analysis over previously published clinical trials. This often requires them to spend significant amounts of time manually going through the large number of studies returned by systems like MEDLINE to find relevant ones and then identify and extract the key data needed to produce evidence reports. This key data is often summarized in a relatively concise way in the form of tables. Automating this process, either fully or even partially, is hindered because the information required to identify relevant studies is often encoded in tables which are beyond the scope of regular text processing and information extraction systems. Thus a labor-intensive data collection process makes performing meta-analyses expensive and results in fewer being done. Figure 1 from [5] clearly shows the huge difference in number of meta-analysis and number of clinical trials published every year.

Web search engines can return tables or web pages containing tables in response to queries such as "US president birthdays" or "temperature change in the Arctic" only if they can understand the intended meaning of column headers and further recognize the relations implicit between columns in a table.

Similarly the MEDLINE search system can do a better job of finding relevant studies when it can realize that the row headers in Figure 2 refer to drug dosages and that the cell values in the columns represent the number of patients cured.

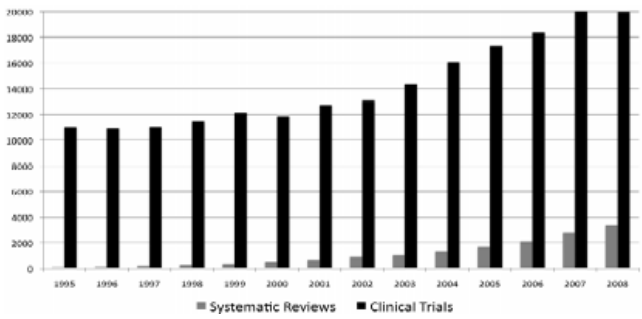



**Figure 1: The number of papers reporting on systematic reviews and meta-analyses is small compared to those reporting on individual clinical trials, as shown in this data from MEDLINE.**

However, the underlying problem with such systems is they fail to capture and understand the intended meaning of tables, often just treating them as free narrative text.

In this paper we focus on the problem of making the intended meaning of tables explicit by representing it as RDF Linked Data [1]. We capture the complete meaning of a table by capturing the meaning of column headers, cell values and relations between columns. The table's meaning is represented Linked RDF data by mapping every column header to a class from a given ontology, linking cell values to existing entities, literal constants or implied measurements and identifying relations between table columns and mapping them to existing ones from the Linked Open Data Cloud [2].

We present an extensible and domain independent framework, which uses background knowledge from LOD and with little or no domain dependence, infers semantics associated with tables. At the core of our framework, is a joint inference module that jointly maps column headers to classes, cell values to entities and identifies relations between table columns. We further incorporate the background knowledge obtained from LOD to improve a message passing scheme used for joint assignments in graphical models. Furthermore, our framework produces Linked Data which reuses existing vocabulary and maps things to URIs instead of strings, producing what Tim Berners-Lee calls 'five-star linked data' [1].

The rest of the paper is organized as follows – Section 2 talks about the process of inferring semantics associated with tables; in Section 3 we give an overview of our domain independent framework and describe the core modules in detail; we present details of our graphical model and inference mechanism using semantics in this section; Section 4 presents evaluation of our framework and discussion; we present related work in Section 5; and we finally conclude discussing future direction in Section 6.

## 2. INTERPRETING A TABLE

It might be tempting to treat table processing and text processing to be similar in nature. After all table too contains text. To understand the difference between text and table processing, consider the example "Freeman A. Hrabowski, III, has served as President of UMBC (The University of Maryland, Baltimore County) since 1992. His research and

**Table 2** *H pylori* eradication rates for each treatment regimen

| | ITT | | PP | |
|---|---|---|---|---|
| | n | % (95% CI) | n | % (95% CI) |
| OAC1W | 240/301 | 79.7 (74.8 to 83.9) | 183/219 | 83.6 (78.1 to 87.9) |
| OAC2W | 246/301 | 81.7 (77 to 85.7) | 185/218 | 84.9 (79.5 to 89.0) |
| OA | 136/305 | 44.6 (39.1 to 50.2) | 96/224 | 42.9 (36.5 to 49.4) |

ITT, intention-to-treat; PP, per protocol; OA, omeprazole 20 mg twice daily and amoxicillin 1 g twice daily and placebo for 2 weeks; OAC1W, omeprazole 20 mg twice daily and amoxicillin 1 g twice daily and clarithromycin 500 mg twice daily for 1 week, followed by omeprazole 20 mg twice daily and placebo for 1 week; OAC2W, omeprazole 20 mg twice daily and amoxicillin 1 g twice daily and clarithromycin 500 g twice daily for 2 weeks.

**Figure 2: Tables in clinical trials literature have characteristics that differ from typical, generic Web tables. They often have row headers well as column headers, most of the cell values are numeric, cell values are often structured and captions can contain detailed metadata. (From [32])**

publications focus on science and math education, with special emphasis on minority participation and performance"[1]. One can understand the intended meaning of the sentence by understanding the meaning on the individual words, whose meaning in turn can be understood with the help of context provided by surrounding words.

Now contrast that to the table in Figure 2 which presents eradication rates for *H-pylori* for different treatment regimes and drug dosages. To comprehend and capture the intended meaning of the table, one would need to understand the row and column headers, which represent different treatment regimes and protocols followed. One would also need to understand the relation between the row and column headers, and map the cell values to literal constants to understand that cell values represent eradication rates. In complex tables like the one in Figure 2, often additional evidence required to understand the meaning of the table in present in text surrounding the table.

It is clear that the intended meaning of tables is often conveyed by the table (column and row) headers, row cell values in the table and also relation between table headers. Often additional context can be obtained from the text surrounding the table. How does one capture this intended meaning ? Consider the leftmost column in the table shown in Figure 3. The column header *City* represents the class and the values *Baltimore*, *Philadelphia*, *New York* and *Boston* are instances of that class. Thus, mapping column headers to existing classes or types and linking the cell values to entities from LOD can capture the meaning of column headers and row cell values. Capturing relationship between table columns can help confirm or deny prior understanding. Consider the strings in the third column of the table in Figure 3. An initial analysis of the column might suggest that they seem to *Politicians*. Additional information that strings in column one represent cities, can help infer that strings in column

[1]source of the text : http://president.umbc.edu/

| *City* | *State* | *Mayor* | *Population* |
|---|---|---|---|
| Baltimore | MD | S.C.Rawlings-Blake | 640,000 |
| Philadelphia | PA | M.Nutter | 1,500,000 |
| New York | NY | M.Bloomberg | 8,400,000 |
| Boston | MA | T.Menino | 610,000 |

**Figure 3: A simple table representing information about cities in United States of America**

three are not only *Politicians* but they are also *Mayors*.

Producing an overall interpretation of a table is a complex task that requires developing an overall understanding of the intended meaning of the table as well as attention to details of choosing the right URIs to represent both the schema as well as instances. We break down the process into following tasks: a) assign every column (or row header) a class label from an appropriate ontology b) link table cell values to appropriate LD entities, if possible c) discover rela- tionships between the table columns and link them to linked data properties d) generate a linked data representation of the inferred data.

# 3. APPROACH

We present an extensible and domain independent framework for inferring the semantics associated with tables and representing it explicitly as Linked Data in Figure 4. Our approach puts emphasis on domain independence and extendibility. The framework should be domain independent because we want a single architecture or system that can deal with tables from a variety of domains, be it from the Web, from medical papers or datasets from sites like www.data.gov with little or no domain dependence. Similarly, we require an extensible framework because it should allow addition of modules to handle practical challenges that may arise when dealing with a very specific set of tables from a given domain that the framework may not be addressing.

Our framework operates as follows. An input table first goes through a set of pre-processing modules which deal with practical challenges. After the initial pre-processing is complete, the table is processed by the query module which generates a ranked list of candidate assignments for column headers, table cells values and relation between table columns, by querying against available sources from the LOD. Once the candidate assignments are generated, the joint assignment module grounded in probabilistic graphical models, jointly assigns values to column headers, cell values and relation between columns with the goal of generating an assignment with which column headers, table cells and relation between columns agree. After the mapping is complete, framework generates triples representing the table as Linked Data. While the goal of our research is to develop to a fully automated framework, achieving the highest level of accuracy demanded for some applications will require human input. Depending upon the application wants to exploit the generated linked data, users may wish to modify the generated interpretation. Our framework will also have an optional module wherein users, if they wish to, can inspect and modify the inferred interpretation after which the data is captured in a knowledge base for other applications to exploit.

While in the following subsections we provide details for each phase in the framework, the focus of this paper is on the implementation and evaluation of two core modules: the *query and rank* module that generates initial candidate assignments for column types, cell values and column relations and the *joint inference module* that iteratively finds the most coherent set of assignments.

## 3.1 Pre–processing

The pre-processing phase will consist of modules to handle number of practical challenges encountered in the process of understanding the semantics of tables such as handling large tables or tables with acronyms, encoded values and literal data in form of numbers and measurements. We envision these pre-processing modules as pluggable modules that can be developed independently and added to the framework without affecting other modules or hampering the flow of the framework. Puranik [19] in his *Masters Thesis* developed and demonstrated one such module which identified whether a column in a table consists of commonly encoded data such as *SSN*, *zip codes*, *phone numbers*, *address* and so on. The purpose of this module is to reduce the load on the more complex joint inference module.

## 3.2 Query and Rank

The *query and rank* module is responsible for generating an initial set of candidate assignments for each of the column headers, cell values and relation between columns and ranking the candidates, with the best possible assignments at the top position. We rely on knowledge sources from the LOD collection for generating candidate classes and entities for column headers and cell values. Our existing framework incorporates data from Wikipedia based knowledge–bases DBpedia [3] and Yago [24]. For most general tables, especially the ones found on the Web, these knowledge sources provide good coverage. Based on the domain to which the tables belong to appropriate additional data sources from the LOD cloud can be selected and incorporated in the framework. Automatically selecting data sources from LOD based on the domain to which the table belongs is an open question for now. For practical purposes, one can expect a human expert to select such data sources from LOD which the framework can query.

*Generating and ranking candidates for row cell values.* We generate an initial set of candidate entities for each table cell value using Wikitology [26, 25], a hybrid knowledge base that combines unstructured and structured information from Wikipedia, DBpedia and Yago. For every cell value/row value in the table, we incorporate the context provided by the column header and rest of the cell values in the given row and query against Wikitology. For example, the query for the string *Baltimore* consists of the query string *Baltimore* and the context for 'disambiguating' *Baltimore* which is provided the column header string *City* and rest of the row values *MD*, *S.C.Rawlings-Blake*, and *640,000*. The details of how the query is formulated and mapped to various fields in Wikitology are presented in [17]. For every query, Wikitology returns a set of matching Wikipedia entities. The query for *Baltimore*, for example, would return Wikipedia entities such as *Baltimore*, *John_Baltimore*, *Baltimore_Ravens* etc. Furthermore for every entity returned, we additionally query DBpedia and Yago to retrieve the en-

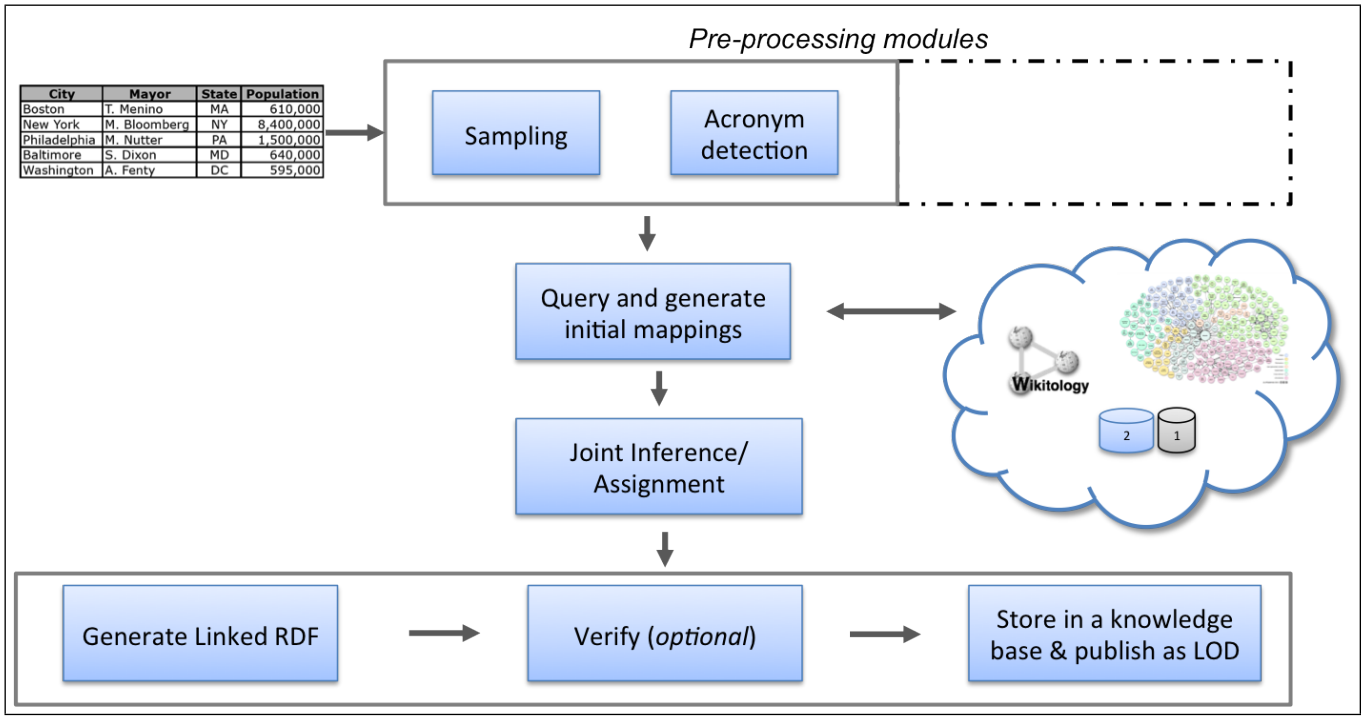



**Figure 4: We are developing a robust domain independent framework for table interpretation that will result in a representation of the information extracted as RDF Linked Open Data.**

tity classes/types. The entity *Baltimore* has DBpedia classes *City, PopulatedPlace, Place* and Yago types *CitiesInMaryland, GeoclassPopulatedPlace* to name a few.

We train and develop an *entity ranker* which uses a machine learning classifier to rank the set of candidate entities associated with each row cell value. We train a classifier to identify whether a specified entity is a correct assignment or not, given the string mention from row cell value and associated features. The general architecture and approach was adapted from one we used in [8]. Based on a set of string similarity features and popularity features, the classifier returns a measure of how likely the given entity URI *John_Baltimore* is correct assignment for a given string mention *Baltimore*.

The string similarity metrics used as features by the classifier include the Levenshtein distance [14], the Dice score [22] between the string mention of row cell value and the entity URI. The notion behind using string similarity metrics is that string mention in the cell and the entity URI are likely to have a high overlap in terms on name. The set of popularity metrics include the predicted page rank [27] of the entity, entity's Wikipedia article length, Wikitology's internal index score for the entity for the specified query and entity's index (or position) in the original ordering returned by Wikitology. The notion behind popularity metrics is that a popular entity is more likely to be the correct disambiguation compared to less popular one.

Given a vector of feature values based on string similarity and popularity metrics, the classifier generates a score of how likely the entity URI is the correct assignment. The *entity ranker* generates this score for every candidate entity associated with a cell value and orders the candidate entity set in decreasing order to generate a ranked list. Thus for the cell *Baltimore* the classifier will generate this score for each of the entities *Baltimore, John_Baltimore, Baltimore_Ravens* and order this set in decreasing order of the score. Every cell value's initial entity assignment is the top ranked entity from its ordered candidate set.

*Generating candidate for column headers.* The candidate classes for every column header are generated based on row cell values in the particular column as described in algorithm 1. Each row cell value is associated with a set of candidate entities and every entity is associated a set of DBpedia and Yago classes. The set of classes associated with a column header is generated by simply taking a union of the set of classes associated with each candidate entity for all the cell values in the particular column. We generate two separate set of candidate classes – one for DBpedia candidate classes and the other for the Yago candidate classes.

*Literal Constants.* We make distinction between string mentions and literal constants such as numerical data in table cell values. We believe that literal constants, unlike their string counterparts, do not represent entities. They often represent values that can be associated with a property or a relation from the LOD cloud. Using regex based techniques, we identify whether a cell value is a literal constant (e.g. numerical data) or string mention. If the cell value is a litral, candidate entities are not generated and the cell is mapped to 'No Annotation'. If all the cell values in a particular column map to literals, we update the column header annotation to 'No Annotation'.

**Algorithm 1** Generate Candidate Classes for Column Header

```
Let Cols be set of columns in a table
for all C in Cols do
    Let 'ColCandidateClasses' be set of candidate classes for C and RowCellValues be the set of cell values in C
    for all r in RowCellValues do
        Let EntitySet be the candidate entity set for r.
        for all Candidate Entity e in EntitySet do
            Let t be set of types associated with e
            Add t to 'ColCandidateClasses'
        end for
    end for
end for
```

## 3.3 Joint Inference

Once the initial sets of candidate assignments are generated, the joint inference module assigns values to column headers, row cell values and identifies relations between the table columns. The result is a representation of the the meaning of the table as a whole. Probabilistic graphical models [12] provide a powerful and convenient framework for expressing a joint probability over a set of variables and performing inference or joint assignment of values to the variables. Probabilistic graphical models use graph based representations to encode probability distribution over a set of variables for a given system. The nodes in such a graph represent the variables of the system and the edges represent the probabilistic interaction between the variables.

We represent a table as Markov network graph in which the column headers and row cell values represent the variable nodes and the edges between them represent their interactions. We choose to model our table - graph as Markov network since it allows to capture interaction between the variables in which the direction of interaction (edge) does not matter. In the case of tables, interaction between the column headers, table cell values and relation between table columns are symmetrical and thus Markov network is better suited for tables.

Figure 5(a) shows interaction between the column headers (represented by $C_i$ where $i \in 1$ to 3) and row cell values (represented by $R_{ij}$ where where $i, j \in 1$ to 3). In a typical well-formed table, each column contains data of a single syntactic type (e.g., strings) that represent entities or values of a common semantic type (e.g., people). For example, in a column of cities, the column header *City* represents the semantic type of values in the column and *Baltimore*, *Boston* and *Philadelphia* are instances of that type. Thus knowing the type (or class) of the column header, influences the decision of the assignment to the table cells in that column and vice-versa. To capture this interaction, we insert an edge between the column header variable and each of the row cell values in that column.

Table cells across a given row are also related. Consider a table cell with a value Beetle. It might be referring to an insect or a car. The next table cell has a value red which is a color. The value in the last table cell is Gasoline, a type of fuel source. All the values considered together, indicate that the row is representing values of a car rather than an insect. Thus to disambiguate a table cell correctly, the context from the rest of table cells in the row should be used. This co-relation when considered between pairs of table cell values between two columns can also be used to identify relation between table columns. To capture this context, we insert edges between all the table cells in a given row.

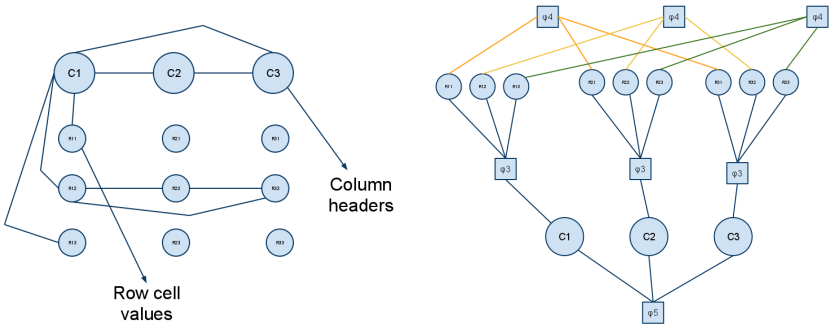
Similar interaction also exist between the column headers. The column header City might suggest that the strings in the columns are cities. However if City appears in the table with other columns which are Basketball players, Coach and Division, we can infer that the column of cities is referring to a team itself an example of metonymy in which the team is referenced by one of its significant properties, the location of it's base. This interaction is captured by inserting edges between column header variables.

To perform any meaningful inference over the graph, one needs to parametrize the graph. We do so by representing the graph in Figure 5(a) as a factor graph as shown in Figure 5(b). The square nodes in the graph represent what are known as '*factor nodes*'. Factor nodes computes and captures affinity or agreement between interacting variables. For example, $\psi_3$ in Figure 5(b) computes agreement between the class assigned to column header and entities linked to the row cell values in that column; $\psi_4$ between row cell values and $\psi_5$ between column headers.

The factor nodes allow the joint inference process to operate. Typical inference algorithms such as Belief Propagation, Message Passing rely on pre–computed joint probability distribution tables stored at each of the factor nodes. For example, the factor node $\psi_3$ for column header variable $C_1$ would store a probability distribution table over the variables $C_1, R_{11}, R_{12}, R_{13}$; i.e. $\psi_3$ would pre–compute and store a joint distribution table over the column header and all row cell values. As the candidate set / possible values that $C_i$ and $R_{ij}$ can be mapped to increase, the size of probability distribution table would rapidly grow. Let us assume that the candidate set for each variable is set to 25; in that case $\psi_3$ associated with the variables $C_1, R_{11}, R_{12}, R_{13}$ would have *390,625* entries in the joint distribution table!

We propose an alternate variation of a inference algorithm which incorporates semantics and background knowledge from LOD to avoid the problem of computing large joint probability distribution tables at factor nodes. Our algorithm, dubbed *'SemMessPass' or 'Semantic Message Passing'* is conceptually similar to the idea of Message Passing schemes. Algorithm 2 gives a high level overview of our proposed Semantic Message Passing algorithm.

The variable nodes in the graph send their current assignment or value to all the factor nodes it is connected to. For example, $R_{11}$ sends its current assignment to factors $\psi_3$ and $\psi_4$. Once the factor nodes receive values from all connected variable nodes, it attempts to compute agreement between the assigned values it receives. Thus, in one of the iterations, $\psi_3$ might receive values *City*, *Baltimore_Ravens*, *Philadelphia*, *New_York* and *Boston*. The goal of $\psi_3$ is to identify if all the assignments agree and identify any outliers. In this case $\psi_3$ will be able to identify that *Baltimore_Ravens* is an




(a) (b)

**Figure 5: (a) This graph represents the interactions between the variables in a simple table. Only some of the connections are shown to keep the figure simple. (b) This factor graph is a parameterized Markov network with nodes for variables and factors.**

**Algorithm 2** *Semantic Message Passing*

1: Let $Vars$ be the set of Variable Nodes and $Factors$ the set of factor nodes in the graph
2: **for all** $v$ in $Vars$ **do**
3: Let $F'$ be set of Factor nodes $v$ is connected to
4: **for all** $f'$ in $F'$ **do**
5: Send current assignment value to $f'$
6: **end for**
7: **end for**
8: **for all** $f$ in $Factors$ **do**
9: Compute agreement between received values
10: Identify variable node that may have sent incorrect/outlier value
11: Send message of "NO–CHANGE" to nodes that are agreement with other nodes as determined by $f$
12: Send message "CHANGE" and characteristic of expected value to nodes that have incorrect assignment as determined by $f$
13: **end for**
14: **for all** $v$ in $Vars$ **do**
15: Let $Messages$ be the set of messages received by $v$
16: If all m $\in$ $Messages$ are NO–CHANGE, do nothing
17: If few or all m $\in$ $Messages$ are CHANGE, update assignment by choosing value from candidate set incorporating characteristics sent by factor nodes
18: **end for**
19: Repeat till convergence

outlier and sends a message of "CHANGE" to $R_{11}$. It also sends characteristics of the value that $R_{11}$ should update to i.e. in this case $\psi_3$ will inform $R_{11}$ to update to an entity of type *City*. For rest of the variable nodes, $\psi_3$ sends a message of "NO–CHANGE". This process will be performed by all factor nodes. Once a variable node receives messages from all the connected factor nodes, it decides whether to update its value or not. If it receives a message of "NO–CHANGE" from all factor nodes, it implies its current assignment is in agreement with others and it need no update its value. If the variable node receives a message of "CHANGE" from some or all factor nodes, it updates it current assignment by taking into consideration the recommended characteristics send by the factor nodes. If the variable nodes update their value, the entire process repeats until convergence or agreement over the entire graph is achieved. A hard convergence metric could be repeat the process until no variable node receives a message of "CHANGE"

Our proposed *Semantic Message Passing* algorithm thus circumvents the problem of computing joint distribution tables at factor nodes, by computing agreement over current assigned values. Furthermore, our scheme, not only proposes to detect individual variable nodes that have incorrect assignment, but it also proposes to send characteristics or *semantics* associated with the value that a variable node should update to. The trick lies in defining powerful factor nodes that can perform such functions. For the purposes of this paper, we implemented and evaluated one key relation from the graph; the factor node $\psi_3$ – the interaction between the column headers and row cell values in each column which jointly maps column header to a LOD class from an ontology and links the cell values to LOD entities.

$\psi_3$ – ***The Column header and row cell value agreement function.*** Algorithm 3 gives an overview of the function that computes agreement between the class to be mapped to column header and entities to be assigned to row cell values in that column. That is agreement between *City*, *Baltimore_Ravens*, *Philadelphia*, *New_York* and *Boston* Recall that at the end of the *query and rank* phase, every row cell value has an initial entity assignment and every column header has a set of candidate classes.

In our current implementation every column header $C_i$ maintains two separate set of candidate classes – one from the Yago ontology and other from the DBpedia ontology. Each row cell value in the given column is mapped to an initial entity $e$ which in itself has its own set of yago and dbpedia classes. The initial entities assigned to row cell values in the column perform a majority voting over Yago and DBpedia class set to pick the top yago and dbpedia class. Each entity votes and increments the score of a class from the candidate set by 1 if the class is present in the class set associated with $e$.

The Yago and DBpedia candidate class sets are ordered by votes, with one at the top having maximum number of 'votes'. $\psi_3$ computes the top score for each of the top classes.

**Algorithm 3** $\psi_3$ Column Header – Row Values Agreement

Let $Yago$ and $DBp$ be set of candidate classes from the Yago and DBpedia ontology for a column $C_i$
Let the initial score for all classes in $Yago$ and $DBp$ be zero.
Let $RowVals$ be the set of row cell values in $C_i$
**for all** $r$ in $RowVals$ **do**
    Let $e$ be the current assigned entity to $r$
    Let $t_y$, $t_d$ be the set of yago and dbpedia classes for $e$
    Majority Vote Score : For all types in $t_y$ that are also present in $Yago$, increment score by 1 for each such class in $Yago$. Similarly, using $t_d$ increment scores for classes in $DBp$
**end for**
**for all** $y$ in $Yago$ **do**
    Get "granularity" score for $y$
**end for**
Order $Yago$ in descending, first by Vote Score and then by granularity score. The yago class at the top of the list is one with maximum votes and also with best "granularity score". Let this class be $top_y$
Order $DBp$ in descending by Vote Score only. The class at the top is one with maximum votes. Let this class be $top_d$
Let $topScore_y$ and $topScore_d$ be scores for top $Yago$ and $DBp$ classes respectively. $topScore = numberOfVotes/numberOfRows$
Check if $topScore_y$ and $topScore_d$ are below threshold.
If scores are below threshold, send message "LOW CONFIDENCE" and "NO–CHANGE" variable nodes
Use $top_y$, $top_d$ if their scores are above threshold in the send message and update process. If either of the class score is above threshold and other is below, check if the classes are "aligned". If the classes are aligned, the class with below threshold is used during update process.
**for all** $r$ in $RowVals$ **do**
    Let $e$ be the current assigned entity to $r$
    If $t_y$, $t_d$ for $e$, do not contain either $top_y$ or $top_d$, send message CHANGE. Send the $top_y$, $top_d$ as the potential classes for Entity $r$ should update to.
    If $t_y$, $t_d$ for $e$, contain either $top_y$ or $top_d$, send message NO–CHANGE
**end for**

The top score is simply equal to numberOfVotesForTopClass /numberOfRows. The Yago classes are further ordered by what we call as 'specificity' or 'granularity' score. Ideally, we want to pick more specific classes (e.g. City) over general classes(e.g. Place) when making an assignment to column headers. Thus, if the mutiple yago classes have received the same number of maximum votes, we use this score as a tie-breaker. We pre–computed specificity scores for all Yago classes. The specificity score is computed by simply dividing the number of instances that belong to the class by the total number of instances and subtracting the result from one. This assigns a higher score to specific classes and a lower score to general classes.

Once the top class(es) are identified and their scores computed, $\psi_3$ checks if they can be used in the process of send message and update process. It checks whether the top scores for the classes are below are certain threshold. If the scores are below a certain threshold, it implies confidence and agreement between row cell values is less and the top classes cannot be relied upon. In such scenarios $\psi_3$ sends a message of "LOW–CONFIDENCE" and "NO–CHANGE" to the variable nodes. $\psi_3$ also maps the column header class to "No Annotation".

If scores for both top yago and dbpedia class are above threshold, $\psi_3$ assigns both the classes to the column header and uses them in the process of send message and update. However if either of the class is below threshold, it checks if the classes are aligned. We define the two classes as aligned if either the DBpedia class is a subclass of the Yago class or vice-versa. The subclass relation between DBpedia and Yago classes is obtained via the PARIS project[2] [23]. If the alignment exists, then ignoring the lower score to either Yago or DBpedia, the $\psi_3$ picks both the classes as the Yago and DBpedia assignments to column header respectively. If the alignment does not exist, the class with the lower score is ignored, and the one with score above threshold is picked as the class for the column header. Once the column header is mapped to class assignments, $\psi_3$ revisits each entity assignment in the column. All row cell value (variable nodes) whose current assigned entity $e$ include the top class(es) in their class set are sent a message of NO–CHANGE; whereas the ones whose entity do contain the top class in their class set are sent a message of CHANGE. These variable nodes are also provided with the top class(es) as semantic /characteristic that their next entity assignment should fulfill.

***Updating the entity annotations for row cell value variables.*** Row cell values that receive a message of CHANGE update their entity assignments. The row cell value variable node picks the next best candidate entity from it's ranked candidate set, that either has one of the top classes in their set of classes. In cases where only the top Yago or only the top DBpedia class is present, the available class is used to update any incorrect assignments.

However, there is an exception to the above process when the candidate set of entities returned by Wikitology for cell value are all low confidence entities. If the index score associated with all the entities is below a certain threshold ($index\_threshold$) it indicates that the entity that can be mapped to the row cell value is not present in the candidate set and perhaps the entity is absent from the knowledge base. In such cases, the algorithm maps the row cell value to "No Annotation". In cases where, the class assignment for column header is "No Annotation", we retain the initial entity assignment. The only update is, in the case of low confidence candidate entities as described above, wherein the row cell value is mapped to 'No Annotation'. In our current implementation, the inference process stops here. Our message passing scheme chooses the the assignments recommended by $\psi_3$ as final assignments. When other factor nodes in the graph are activated the inference process would define a new convergence metric.

# 4. EVALUATION

We divide the evaluation section into two parts – the first section discusses experiments and evaluations for column

[2]http://webdam.inria.fr/paris/

| | |
|---|---|
| *Column header Annotation* | *0.6486* |
| *Row Cell Value Annotation* | *0.7591* |

**Figure 6: $F_1$ score for column header annotations and Accuracy for Row Cell Value annotations**

header and row cell value annotations; the second section talks about the performance of the *entity ranker*.

## 4.1 Column Header & Cell Value Annotations

For the purposes of evaluation, we use a subset of 80 tables extracted from Wikipedia articles, belonging to the 'wiki-links' dataset obtained from [15]. The dataset is labeled and every cell value is either linked to an entity from the Yago knowledge base or 'NA' (which represents No Annotation) in cases where the entity does not exists. We use their entity annotations as ground truth for evaluating our framework's entity annotation part. The dataset also provide annotations for column headers, by mapping column headers either to Yago or Wordnet classes. We choose not to use their class annotations for our evaluation purpose, since our current setup includes classes from the DBPedia and Yago ontologies.

Our existing knowledge source used for generating candidate classes and entities is largely based on Wikitology and DBpedia. The version of Wikitology we use was built from early 2010 dump of Wikipedia and is likely to miss entities that may be added Wikipedia post that time period. We obtain DBpedia classes and Yago classes associated with DBpedia / Wikipedia entities via the DBpedia dumps available at http://wiki.dbpedia.org/Downloads38. In the evaluation results described below the parameter *index_threshold* is set to 10 and the top class score threshold is set to 0.5.

We compute accuracy or the percentage of correctly linked entities in the evaluation of mapping row cell values to entities. We compare our annotation against the ground truth in our dataset and consider our annotation to be correct when it matches the ground truth. For column header annotations we report $F_1$ score. For every column, our framework produces top three classes from the Yago candidate classes as well top three from the DBpedia candidate classes. We let human judges evaluate the produced output. We divided the dataset of 80 tables between four different judges. Judges marked a predicted class label 'Good' if they found the prediction to be correct, 'Okay' if they found the predicted label correct, but not specific enough and Incorrect if they thought the predicted label was incorrect. Recollect, in cases where the column contained all literals like numerical constants, we choose to predict NA and such predictions considered correct by the judges. However if we predicted a class for column instead of NA, judges considered such predictions to be incorrect. Similar to strategy in [29], to compute precision we assign a score of 1 if a human judge marked our predicted class label as good, 0.5 if the judge marked it as okay and 0 if the judge marked it as incorrect. Similarly while computing recall, the predicted class label was assigned as score of 1 if it was marked good or okay and 0 if it was marked incorrect.

For every column header, our system attempted to produce

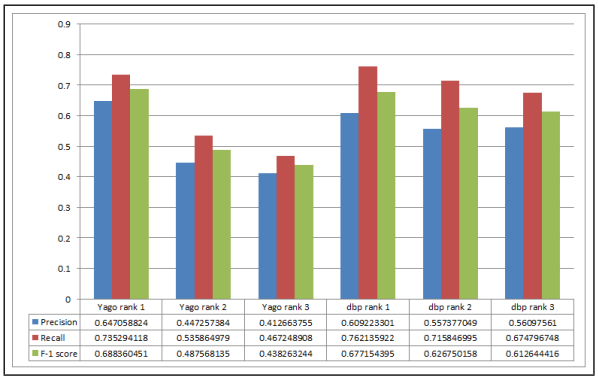



**Figure 7: Individual $F_1$ score for the top three Yago and DBpedia classes.**

the top three Yago and top three DBpedia classes, which is not always possible, since some instances on DBpedia are tagged with only Yago classes or only DBpedia classes. Over the dataset of 80 tables, our system was able to produce a combination of 1203 class labels (which included NA as well). Our human judges marked 522 classes to be good, 259 to be okay and 436 to be incorrect. We obtain an $F_1$ of *0.6486* over 1203 classes. While a direct comparison between our evaluation and [15] is not possible due to difference in datasets for column header annotation as well as evaluation strategy, we do note that our $F_1$ score is better previously reported score of *0.56* [15] and slightly lower than *0.67* reported in [29]. We also compute separate $F_1$ score for each of the top three yago and dbpedia classes (Figure 7). The $F_1$ scores for our top–ranked Yago and top–ranked DBpedia classes across our dataset are better than any previously reported scores.

We now present results for entity linking. The dataset of 80 tables had in all 3981 cell values which could be either linked to LOD entities or mapped to NA. Our framework linked 3022 entities correctly obtaining an accuracy of 75.91 %. While computing accuracy, we haven't accounted for the cases, where the ground truth is an entity missing from our knowledge base and our framework may have correctly predicted as NA and the reverse case where ground truth is NA, but our framework may have correctly predicted an entity. We expect the entity annotation accuracy to increase when these cases are accounted for.

## 4.2 Entity Ranker

As described in the approach section, the *entity ranker* consists of a machine classifier model that generates a metric of how likely the given entity is correct assignment for a given string mention. The training and test datasets were generated using the ground truth for entity annotations from our dataset of 80 tables. For every string mention in the table, we queried Wikitology as described in Section 3.2 to generate a set of candidate entities. For each pair of string mention and candidate entity, feature values for the string similarity and popularity metrics were generated. A class label of 'Yes' (or 1) was assigned if the candidate entity was the correct assignment (available via ground truth in the dataset) else a class label of 'No' (or 0) was assigned. The training set included 600 instances and was evenly split with

| *Class* | *Precision* | *Recall* | *F-Measure* |
|---|---|---|---|
| *Yes* | 0.959 | 0.849 | 0.901 |
| *No* | 0.871 | 0.966 | 0.916 |

**Figure 8: Precision, Recall and F-Measure for the Naive Bayes model**

300 positive (i.e label = yes) and 300 negative (i.e label = no) instances. The test set included in all 681 instances with 331 positive and 350 negative instances.

The model used in the entity ranker is trained using Naive Bayes. We choose to use Naive Bayes because the features (string similarity and popularity metrics) are fairly independent of each other. The model was trained over the training set described above using default parameters and its performance was evaluated against the test set. Out of the 681 instances, the model was able to correctly classify 619 instances with an accuracy of *90.89 %*. The precision, recall and F-measure are presented in Figure 8.

# 5. RELATED WORK

Our work is closely related to two threads of research, one that focuses on pragmatically generating RDF and linked data from databases, spreadsheets and CSV files and a more recent one that addresses understanding and inferring the implicit semantics of tables.

Several systems have been implemented to generate semantic web data from databases [21, 28, 18], spreadsheets [10, 13] and CSV files [7]. All are manual or only partially automated and none have focused on automatically generating *linked* RDF data for the entire table. These systems have mainly focused on relational databases where the schema is available or on simple spreadsheets. In the domain of open government data, [7] presents techniques to convert raw data (CSV, spreadsheets) to RDF. However the generated RDF data does not use existing classes or properties for column headers, nor does it link cell values to entities from the LOD cloud. To generate a richer, enhanced mappings, users will need to manually specify a configuration file.

The key shortcomings of these systems are twofold: they rely heavily on users who must be Semantic Web experts and they do not produced *linked data*. These systems do not automatically link classes and entities generated from their mapping to existing Semantic Web resources – their output turns out to be just *raw string data* represented as RDF, instead of fully linked RDF. Figure 9 shows a part of RDF representation of dataset 1425 from *data.gov* [6]. The property names in the representation are column headers from the raw dataset and the values of the properties represent row values for the respective columns.

```
<rdf:Description rdf:about="#entry1">
<value>6444</value>
<label>Number of Farms</label>
<group>Farms with women principal operators</group>
<county_fips>000</county_fips>
<state_fips>01</state_fips>
<state>Alabama</state>
<rdf:type rdf:resource="http://data-gov.tw.rpi.edu/2009
/data-gov-twc.rdf#DataEntry"/>
</rdf:Description>
```

**Figure 9: A portion of the RDF representation from dataset 1425 - Census of Agriculture Race, Ethnicity and Gender Profile Data from data.gov.**

The representation fails to use existing vocabulary terms to annotate the raw data and most of the column headers are mapped to properties local to the RDF file. Mapping column headers to classes and properties from the LOD cloud, provides richer description as compared to the local properties. Such a representation often uses string identifiers for table cell values instead of linking them to existing entities in the LOD cloud. Linking the string cell values can further enrich the semantic representation of the data. Our framework will link and reuse existing classes, properties and entities with dereferenceable URIs from the LOD cloud. Our goal is to generate linked data in a form which is identified as "five star" by Tim Berners-Lee [1].

Early work in table understanding focused on extracting tables from documents and web pages [11, 9] with more recent research attempting to understand their semantics. Wang et al. [30] began by identifying a single 'entity column' in a table and, based on its values and rest of the column headers, associates a concept from the Probase [31] knowledge base with the table. Their work does not attempt to link the table cell values or identify relations between columns. Ventis et al. [29] associate multiple class labels (or concepts) with columns in a table and identify relations between the 'subject' column and the rest of the columns in the table. Both the concept identification for columns and relation identification is based on maximum likelihood hypothesis, i.e., the best class label (or relation) is one that maximizes the probability of the values given the class label (or relation) for the column. Their work also does not attempt to link the table cell values. Limaye et al. [15] use a graphical model which maps every column header to a class from a known ontology, links table cell values to entities from a knowledge-base and identifies relations between columns. They rely on Yago for background knowledge.

The core of our framework is a probabilistic graphical model that captures a much richer semantics, including relation between column headers as well relation between entities across a given row. Our model has a single 'factor' node to capture relation between column header and strings in the column, which makes it possible to deal with missing values (e.g., absent column header).

Current systems for interpreting tables rely on semantically poor and possibly noisy knowledge-bases and do not attempt to produce a complete interpretation of a table. None of the current systems propose or generate any form of linked data from the inferred meaning. The work mentioned above will work well with string based tables but we know of no systems that interpret columns with numeric values and use the results as evidence in the table interpretation. Doing so is essential for many domains, including medical research.

# 6. CONCLUSIONS

Generating an explicit representation of the meaning implicit in tabular data will support automatic integration and more accurate search. Clues for a table's intended meaning are present in column and row headers, cell values, implicit relations between columns, and any descriptive text. We described general techniques grounded in graphical models and probabilistic reasoning to infer a tables meaning relative to a knowledge base of general and domain-specific knowledge expressed in the Semantic Web language OWL. We represent a table's meaning as a graph of OWL triples where the columns have been mapped to classes, cell values to literals, measurements, or knowledge-base entities and relations to triples. Our joint inference model incorporates semantics and background knowledge from the LOD to improve on existing message passing schemes and evaluation shows promising results. An immediate future goal would be extending the implementation of our graphical model. Converting tabular data to RDF or high quality linked data has been long standing challenge for the Semantic Web community. We believe our extendable and domain independent framework can address and overcome the existing challenges and in this paper, we presented a building block for realizing such this framework.

# 7. REFERENCES


[1] T. Berners-Lee. Linked data. http://www.w3.org/DesignIssues/LinkedData.html, July 2006.

[2] C. Bizer. The emerging web of linked data. *IEEE Intelligent Systems*, 24(5):87–92, 2009.

[3] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. Dbpedia - a crystallization point for the web of data. *Journal of Web Semantics*, 7(3):154–165, 2009.

[4] M. J. Cafarella, A. Y. Halevy, Z. D. Wang, E. Wu, and Y. Zhang. Webtables: exploring the power of tables on the web. *PVLDB*, 1(1):538–549, 2008.

[5] A. Cohen, C. Adams, J. Davis, C. Yu, P. Yu, W. Meng, L. Duggan, M. McDonagh, and N. Smalheiser. Evidence-based medicine, the essential role of systematic reviews, and the need for automated text mining tools. In *Proc. 1st ACM Int. Health Informatics Symposium*, pages 376–380. ACM, 2010.

[6] Dataset 1425 - Census of Agriculture Race, Ethnicity and Gender Profile Data. http://explore.data.gov/Agriculture/Census-of-Agriculture-Race-Ethnicity-and-Gender-Pr/yd4n-fk45. 2009.

[7] L. Ding, D. DiFranzo, A. Graves, J. R. Michaelis, X. Li, D. L. McGuinness, and J. A. Hendler. Twc data-gov corpus: incrementally generating linked government data from data.gov. In *Proc 19th Int. Conf. on the World Wide Web*, pages 1383–1386, New York, NY, USA, 2010. ACM.

[8] M. Dredze, P. McNamee, D. Rao, A. Gerber, and T. Finin. Entity disambiguation for knowledge base population. In *COLING*, pages 277–285, 2010.

[9] D. W. Embley, D. P. Lopresti, and G. Nagy. Notes on contemporary table recognition. In *Document Analysis Systems*, pages 164–175, 2006.

[10] L. Han, T. Finin, C. Parr, J. Sachs, and A. Joshi. RDF123: from Spreadsheets to RDF. In *Proc. 7th Int. Semantic Web Conf.* Springer, October 2008.

[11] M. Hurst. Towards a theory of tables. *IJDAR*, 8(2-3):123–131, 2006.

[12] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[13] A. Langegger and W. Wob. Xlwrap - querying and integrating arbitrary spreadsheets with SPARQL. In *Proc. 8th Int. Semantic Web Conf.*, October 2009.

[14] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. Technical Report 8, Soviet Physics Doklady, 1966.

[15] G. Limaye, S. Sarawagi, and S. Chakrabarti. Annotating and searching web tables using entities, types and relationships. In *Proc. 36th Int. Conf. on Very Large Databases*, 2010.

[16] V. Mulwad, T. Finin, and A. Joshi. A Domain Independent Framework for Extracting Linked Semantic Data from Tables. In *Search Computing - Broadening Web Search*, pages 16–33. Springer, July 2012. LNCS volume 7538.

[17] V. Mulwad, T. Finin, Z. Syed, and A. Joshi. Using linked data to interpret tables. In *Proc. 1st Int. Workshop on Consuming Linked Data*, Shanghai, 2010.

[18] S. Polfliet and R. Ichise. Automated mapping generation for converting databases into linked data. In *Proc. 9th Int. Semantic Web Conf.*, November 2010.

[19] N. Puranik. A specialist approach for classification of column data. Master's thesis, University of Maryland, Baltimore County, August 2012.

[20] D. Sackett, W. Rosenberg, J. Gray, R. Haynes, and W. Richardson. Evidence based medicine: what it is and what it isn't. *Bmj*, 312(7023):71, 1996.

[21] S. S. Sahoo, W. Halb, S. Hellmann, K. Idehen, T. Thibodeau Jr, S. Auer, J. Sequeda, and A. Ezzat. A survey of current approaches for mapping of relational databases to rdf. Technical report, W3C, 2009.

[22] G. Salton and M. J. Mcgill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986.

[23] F. M. Suchanek, S. Abiteboul, and P. Senellart. PARIS: Probabilistic Alignment of Relations, Instances, and Schema. *PVLDB*, 5(3):157–168, 2011.

[24] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A Core of Semantic Knowledge. In *16th Int. World Wide Web Conf.*, New York, 2007. ACM Press.

[25] Z. Syed. *Wikitology: A Novel Hybrid Knowledge Base Derived from Wikipedia*. PhD thesis, University of Maryland, Baltimore County, August 2010.

[26] Z. Syed and T. Finin. *Creating and Exploiting a Hybrid Knowledge Base for Linked Data*. Springer, April 2011.

[27] Z. Syed, T. Finin, V. Mulwad, and A. Joshi. Exploiting a Web of Semantic Data for Interpreting Tables. In *Proceedings of the Second Web Science Conference*, April 2010.

[28] K. N. Vavliakis, T. K. Grollios, and P. A. Mitkas. Rdote - transforming relational databases into semantic web data. In *Proc. 9th Int. Semantic Web Conf.*, November 2010.

[29] P. Venetis, A. Halevy, J. Madhavan, M. Pasca,

W. Shen, F. Wu, G. Miao, and C. Wu. Recovering semantics of tables on the web. In *Proc. 37th Int. Conf. on Very Large Databases*, 2011.

[30] J. Wang, B. Shao, H. Wang, and K. Q. Zhu. Understanding tables on the web. Technical report, Microsoft Research Asia, 2011.

[31] W. Wu, H. Li, H. Wang, and K. Zhu. Towards a probabilistic taxonomy of many concepts. Technical report, Microsoft Research Asia, 2011.

[32] R. Zagari, G. Bianchi-Porro, R. Fiocca, G. Gasbarrini, E. Roda, and F. Bazzoli. Comparison of 1 and 2 weeks of omeprazole, amoxicillin and clarithromycin treatment for helicobacter pylori eradication: the hyper study. *Gut*, 56(4):475, 2007.