# Semantic Message Passing for Generating Linked Data from Tables*

Varish Mulwad, Tim Finin and Anupam Joshi

University of Maryland, Baltimore County
Baltimore, MD 21250 USA
{varish1,finin,joshi}@cs.umbc.edu

**Abstract.** We describe work on automatically inferring the intended meaning of tables and representing it as RDF linked data, making it available for improving search, interoperability and integration. We present implementation details of a joint inference module that uses knowledge from the linked open data (LOD) cloud to jointly infer the semantics of column headers, table cell values (e.g., strings and numbers) and relations between columns. The framework generates linked data by mapping column headers to classes, cell values to LOD entities (existing or new) and by identifying relations between columns.e also implement a novel *Semantic Message Passing* algorithm which uses LOD knowledge to improve existing message passing schemes. We evaluate our implemented techniques on tables from the Web and Wikipedia.

**Keywords:** Tables, Semantic Web, Linked Data, Graphical Models

## 1 Introduction

Tables are an integral part of documents, reports and Web pages, compactly encoding important information that can be difficult to express in text. Table-like structures outside documents, such as spreadsheets, CSV files, log files and databases, are widely used to represent and share information. Tables are important to a number of domains including the Web, healthcare, e-science and public policy. Google study [2] found more than 150 million high quality relational tables on the Web. Many governments share public data useful to citizens and businesses as tables. The U.S. government's data sharing website, for example, had nearly 400,000 such datasets as of September 2012. Medical researchers can asses treatment efficacy via a meta-analysis of previously published clinical trials, often using systems like MEDLINE[1] to find relevant articles and extract key data, which is typically summarized in tables, like the one in Figure 1.

Integrating and searching over this information benefits from a better understanding of its intended meaning, a task with several unique challenges. The

* This is an extended version of a paper with the same title that appears in the Proceedings of the 2013 International Semantic Web Conference.

[1] http://nlm.nih.gov/bsd/pmresources.html

very structure of tables which adds value and makes it easier for human understanding also makes it harder for machine understanding. Web search engines, for example, perform well when searching over narrative text on the Web, but poorly when searching for information embedded in tables in HTML documents.

We might interpret tables using proven NLP techniques; after all, tables also contain text. We understand the meaning of a sentence by understanding the meaning of the individual words, which in turn are understood using grammatical knowledge and the context provided by the surrounding text. Contrast that with a table like Figure 1, where uncovering its meaning requires interpreting the row and column headers, the relation between them and mapping cell values to appropriate measurements.

**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.

Fig. 1: Tables in clinical trials reports [22] often have both row and column headers, contain numerical data and have captions with critical metadata.

The intended meaning of tables is strongly suggested by the column and row headers, cell values and relations between the columns. Additional context can often be found in a caption or other text near the table. How does one capture this intended meaning? Consider the leftmost column in the table shown in Figure 2. The column header *City* represents the class and the values *Baltimore*, *Philadelphia*, *New York* and *Boston* are instances of that class. Capturing the relationships between table columns can help confirm or deny prior understanding. Consider the strings in the third column of the table in Figure 2. An initial analysis of the column might suggest that they refer to *Politicians*. Additional information that strings in column one represent cities, can help confirm that they are not only *Politicians* but also *Mayors* of the cites mentioned in the first column.

| *City* | *State* | *Mayor* | *Population* |
|---|---|---|---|
| Baltimore | MD | S.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 |

Fig. 2: A table with information about U.S. cities.

Our goal is to encode a table as RDF linked data, mapping columns to appropriate classes, linking cell values to entities, literal constants or implied measurements, identifying relations between columns and asserting appropriate RDF triples. We describe an extensible, domain-independent framework to do this using background knowledge from an LOD resource. A novel feature is the incorporation of semantic knowledge in the message passing algorithm used for joint assignments in a graphical model.

Producing linked data representation is a complex task that requires developing an overall understanding of the intended meaning of the table as well as choosing the right URIs to represent its schema and instances. We decompose it as follows: (1) assign column (and/or row) headers classes from an appropriate ontology, (2) link cell values to literals or entities (creating them as necessary), (3) discover relations between table columns and add properties to represent

them, and (4) generate a linked data representation. We describe our approach to these tasks and an evaluation of the results in the remainder of the paper.

## 2 Approach

Figure 3 shows our extensible and domain independent framework for inferring the meaning of a table and representing it explicitly as linked data. An input table first goes through a *preprocessing phase* with modules to handle a number of pragmatic issues, such as sampling rows from large tables and recognizing and expanding acronyms and stylized literal values (e.g., phone numbers). These modules can be developed independently and added to the framework without affecting others or hampering the workflow. Puranik [14], for example, developed modules to identify whether a column in a table consists of commonly encoded data such as *SSN*, *zip codes*, *phone numbers* and *addresses*.

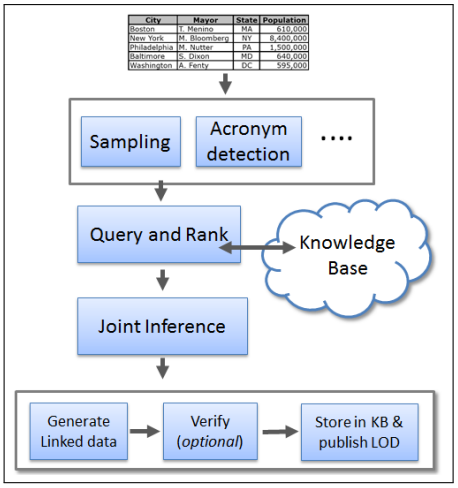


Fig. 3: Our extensible and domain independent framework relies on a *joint inference* module to generate a representation of the the meaning of the table as a whole.

A table is then processed by the *query and rank* module which queries the background LOD sources to generate initial ranked lists of candidate assignments for column headers, cell values and relations between columns. Once candidate assignments are generated, the *joint inference* component uses a probabilistic graphical model to capture the correlation between column headers, cell values and column relations to make class, entity and relation assignments. After the mapping is complete, linked data triples are produced. Although our goal is to develop a fully automated system achieving a high level of accuracy, we recognize that practical systems will benefit from or even require human input. Future work is planned to allow users to view the interpretation and give feedback and advice if it is incorrect.

While we presented a brief sketch of our framework previously [11], the contributions of this paper include (1) an implementation of the graphical model with improved factor nodes, (2) enhancements to our novel *Semantic Message Passing* algorithm and a detailed description of its implementation, and (3) a thorough evaluation. The rest of the section consists of a brief review of the *query and rank* module followed by details of the *joint inference* module, which is the key focus of this paper.

### 2.1 Query and Rank

The *query and rank* module generates an initial ranked list of candidate assignments for the column headers, cell values and column relations using data from

DBpedia [1], Yago [17] and Wikitology [18]. For most of the general tables, especially ones found on the Web, these knowledge sources provide good coverage. Additional LOD data sources can be selected and incorporated, automatically or manually, based on the table's domain.

**Generating and ranking candidates for cell values.** We generate an initial set of candidate entities for each cell value using Wikitology, a hybrid knowledge base combining unstructured and structured information from Wikipedia, DBpedia and Yago. The contents of the column header and other row values are used as context when querying Wikitology. The query for *Baltimore*, for example, consists of the query string *Baltimore* and the context data *City*, *MD*, *S.C.Rawlings-Blake*, and *640,000* [12]. Wikitology returns ranked lists of entities and classes, which for *Baltimore*, include the entities *Baltimore*, *John_Baltimore* and *Baltimore_Ravens* along with DBpedia classes *City*, *PopulatedPlace* and *Place* and Yago types *CitiesInMaryland* and *GeoclassPopulatedPlace*. An *entity ranker* then re-ranks a cell's candidates entities using an approach adapted from [4] and features from [12] to return a measure of how likely the given entity (e.g., *John_Baltimore*) is the correct assignment for the string mention (e.g., *Baltimore*).

**Generating candidates for columns.** Initial candidate classes for a column are generated from its cell values, each of which has a set of candidate entities, which in turn have sets of DBpedia and Yago classes. The column's potential classes is just the union of the classes from the its cells. We generate two separate set of candidate classes – one for DBpedia classes and another for Yago classes.

**Generating candidate relations between columns.** Identifying relations between table columns is an important part of table understanding and is modeled by finding appropriate predicates from the reference LOD's ontologies (e.g., DBpedia). We generate candidate relations for every pair of columns in the table, based on the cell value pairs in the respective columns. Each cell value has a set of candidate entities, which in turn may be linked to other entities in the reference LOD resources. For example, the DBpedia entities *Baltimore* and *Maryland* are linked via the predicates *isPartOf* and *subdivisionName*.

We use the links between pairs of entities to generate candidate relations. For a pair of cell values in the same row between the two columns, the candidate entity sets for both cells are obtained. For each possible pairing between the entities in both the candidate sets, we query Yago and DBpedia, to obtain relation in either direction i.e. *entityrow1 someproperty1 entityrow2* and *entityrow2 someproperty2 entityrow1*. This gives us a candidate set between pair of row cell values. The candidate relation set for the entire column pair is generated by taking a union of the set of candidate relations between individual pairs of row cell values. Thus for example, the candidate relations between column *City* and column *State* might include *isPartOf, capitalCity, bornIn* etc. Again, we generate two sets of candidate relations, one from DBpedia and the other from Yago.

**Literal Constants.** We use a regular expression to distinguish string mentions, which probably refer to entities, and literal constants such as numbers and measurements, which probably do not. If the cell value is a literal constant,

candidate entities are not generated and the cell is mapped to NO-ANNOTATION. If all the cells in a column are literals, we update the column header annotation to NO-ANNOTATION.

### 2.2 Joint Inference

Once the initial sets of candidate assignments are generated, the joint inference module assigns values to columns and row cell values and identifies relations between the table columns. The result is a representation of the meaning of the table as a whole. Probabilistic graphical models [8] 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 a Markov network graph in which the column headers and cell values represent the variable nodes and the edges between them represent their interactions. The edges in a Markov network graph are undirected because the interactions between the variables are symmetrical. In the case of tables, interactions between the column headers, table cell values and the relation between table columns are symmetrical and thus a Markov network is well suited for tables.

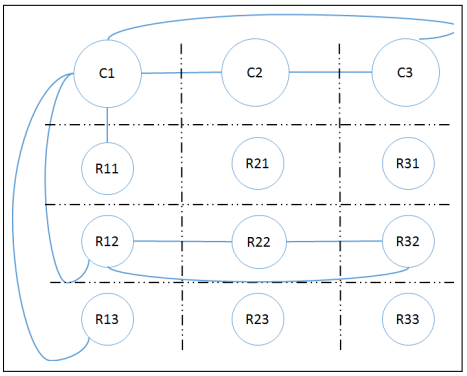


Fig. 4: This graph represents the interactions between the variables in a simple table. Only some of the connections are shown to keep the figure simple.

Figure 4 shows interaction between the column headers (represented by $C_i$ where $i \in 1$ to 3) and cell values (represented by $R_{ij}$ 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 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 cell values in that column.

Table cells across a given row are also related. Consider a table cell with a value *Beetle*, which might refer to an *insect* or a *car*. Suppose an adjacent cell has a string value *red*, which is a reference to a *color*, and another cell in the same row has the string value *Gasoline*, which is a type of *fuel source*. As a collection, the cell values suggest that the row represents values of a *car* rather than an

*insect*. Thus, the interpretation of each cell is influenced by the interpretation of the others in its row. This co-relation when considered between pairs of table cell values between two columns can also be used to identify relations between table columns. To capture this context, we insert edges between all the table cells in a given row.

Similar interactions exist between the column headers. By itself, the column header *City* suggests that column's cells might refer to city instances. However, if the other columns appear to refer to basketball players, coaches and basketball divisions, we can infer that the cities column refers to a team itself. This is an example of metonymy, in which an entity (i.e., the team) is referenced by one of its significant properties (i.e., the location of its base). This interaction is captured by inserting edges between column header variables.

To perform any meaningful inference over the graph, it must be parameterized. We do so by representing the graph in Figure 4 as a factor graph as shown in Figure 5. The graph's square nodes represent what are known as '*factor nodes*', which compute and capture affinity or agreement between interacting variables. For example, $\psi_3$ in Figure 5 computes the agreement between the class assigned to column header and entities linked to the cell values in that column; $\psi_4$ between row cell values for a given pair of columns and $\psi_5$ between column headers.

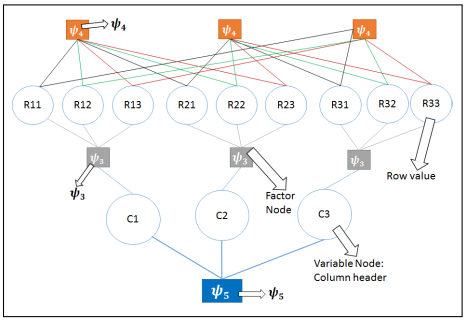


Fig. 5: This factor graph is a parameterized Markov network wherein the square nodes represent factor nodes.

**Semantic Message Passing.** Factor nodes allow the joint inference process to operate. Typical inference algorithms such as *belief propagation* and *message passing* rely on pre-computed joint probability distribution tables (PDTs) stored at the factor nodes. For example, the factor node $\psi_3$ for column header variable $C_1$ would store a PDT over the variables $C_1, R_{11}, R_{12}, R_{13}$; i.e. $\psi_3$ would pre–compute and store a PDT over the column header and all row cell values. As the size of the candidate set of values that $C_i$ and $R_{ij}$ can be mapped to increases, the size of the PDT will rapidly grow. Assuming that the size of the candidate set for a variable is 25, $\psi_3$ associated with the variables $C_1, R_{11}, R_{12}, R_{13}$ would have *390,625* entries in the joint PDT!

We implement a variation of an inference algorithm which incorporates semantics and background knowledge from LOD to avoid the problem of computing large joint PDTs at factor nodes. Our *Semantic Message Passing* algorithm is conceptually similar to the idea of Message Passing schemes. Algorithm 1 gives a high level overview of our '*Semantic Message Passing*' algorithm.

The variable nodes in the graph send their current assignment to the factor nodes to which they are connected. For example, $R_{11}$ sends its current assignment to factor nodes $\psi_3$ and $\psi_4$. Once the factor nodes receive values from all connected variable nodes, they compute agreement between the values. Thus, in

**Algorithm 1** *Semantic Message Passing*

Let $Vars$ be the set of variable nodes and $Factors$ be the set of factor nodes in the graph.
**for all** $v$ in $Vars$ **do**
  Let $F'$ be the set of factor nodes $v$ is connected to.
  **for all** $f'$ in $F'$ **do**
    $v$ sends its current assignment (value) to $f'$.
  **end for**
**end for**
**for all** $f$ in $Factors$ **do**
  Compute agreement between the received assignments.
  Identify variable nodes that may have sent an incorrect assignment.
  Send a NO-CHANGE message to all nodes that are in agreement, as determined by $f$.
  Send CHANGE message and characteristic of expected assignment to all the nodes that have an incorrect assignment, as determined by $f$.
**end for**
**for all** $v$ in $Vars$ **do**
  Let $Messages$ be the set of messages received by $v$.
  If all $m \in Messages$ are NO-CHANGE, do nothing.
  If few or all $m \in Messages$ are CHANGE, update the current assignment by choosing a new one from the candidate set which satisfies the characteristics sent by the factor nodes.
**end for**
Repeat till convergence.

one of the iterations, $\psi_3$ might receive values *City*, *Baltimore_Ravens*, *Philadelphia*, *New_York* and *Boston*. The goal of $\psi_3$ is to determine if all the assignments agree and, if not, identify the outliers.

In this case $\psi_3$ identifies *Baltimore_Ravens* as an outlier and sends a CHANGE message to $R_{11}$, together with its semantic preferences for a new, alternate value that $R_{11}$ might produce. In our example, $\psi_3$ informs $R_{11}$ of its preference for update to an entity of type *City*. To the rest of the variable nodes, $\psi_3$ sends a NO-CHANGE message. This process is performed by all factor nodes. Once a variable node receives messages from all of its connected factor nodes, it decides whether to update its value or not.

If it receives a message of NO-CHANGE from all factor nodes, its current assignment is in agreement with the others and it need not update its assignment. If it receives a CHANGE message from some or all factor nodes, it updates its current assignment, taking into consideration the semantic preferences provided by the factor nodes. The entire process repeats until convergence, i.e., agreement over the entire graph is achieved. A hard convergence metric could be to repeat the process until no variable node receives a CHANGE message.

Our *Semantic Message Passing* algorithm thus circumvents the problem of computing joint PDTs at factor nodes by computing agreement over current assigned values. Furthermore, our scheme not only detects individual variable

nodes that have incorrect assignments, but provides the nodes with guidance on the characteristics or *semantics* associated with the value that a variable node should update to. This capability requires defining semantically-aware factor nodes that can perform such functions. In this paper, we describe our implementation of factor nodes $\psi_3$ and $\psi_4$ and the process by which a variable node updates its values based on the messages received and our metric for graph convergence.

$\psi_3$ – **Column header and row cell value agreement function.** Algorithm 2 gives an overview of the column header and row cell value agreement function. The $\psi_3$ factor node computes agreement between the class assigned to the column and the entities assigned to its cell values. For example, agreement between the column assigned type *City* and candidate cell assignments *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 sets of candidate classes – one from Yago's classes and the other from DBpedia's. Each cell value in a 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 a column's cell values perform a majority voting over the 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. $\psi_3$ computes the top score for each of the top classes. The top score is simply equal to the number of votes for the top class divided by the number of rows in the column. Ideally, we want to pick more specific classes (e.g., City) over general classes (e.g., Place) when making an assignment to the column headers. Thus, if multiple Yago classes get voted as top class, we use a 'granularity' score as tie-breaker. The 'granularity' 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, $\psi_3$ determines if they can be used in the process of identifying cell values with incorrect assignments. It checks whether the top scores for the classes are below a certain threshold. If so, it implies lower confidence and agreement between row cell values and that 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 and also maps the column header class to NO-ANNOTATION.

If scores for both the top Yago and DBpedia classes are above the threshold, $\psi_3$ assigns both the classes to the column header and uses them in the process of identifying its cell values with incorrect assignments. However if either 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.

**Algorithm 2** $\psi_3$ – Column Header – Row Values Agreement

1: Let $Yago$ and $DBp$ be the column header $C_i$'s candidate class sets from Yago & DBpedia respectively and $RowVals$ be the set of row cell values in $C_i$.
2: **for all** $r$ in $RowVals$ **do**
3: Let $e$ be the currently assigned entity to $r$ and $t_y$, $t_d$ be the set of Yago and DBpedia classes for $e$.
4: Majority Vote Score: For each class in $Yago$, increment the score by 1 if it is present in $t_y$. Similarly, using $t_d$, increment the scores for classes in $DBp$.
5: **end for**
6: **for all** $y$ in $Yago$ **do**
7: Get the "granularity" score for $y$.
8: **end for**
9: Order the classes in $Yago$ in descending, first by vote scores and then by granularity scores. Let $top_y$ be the Yago class with maximum vote score and best "granularity score".
10: Order the classes in $DBp$ in descending by vote scores. The class at the top is the one with maximum votes. Let this class be $top_d$.
11: Let $topScore_y$ and $topScore_d$ be scores for $top_y$ and $top_d$ respectively, where $topScore = numberOfVotes/numberOfRows$
12: If the scores are below threshold, send LOW-CONFIDENCE and NO-CHANGE messages to all the row cell values in $RowVals$. Update $C_i$'s class annotations to NO-ANNOTATION.
13: If the scores are above threshold, update $C_i$'s class annotations to $top_y$ and $top_d$. Send appropriate CHANGE or NO-CHANGE messages to row cell values in $RowVals$ as shown next.
14: **for all** $r$ in $RowVals$ **do**
15: Let $e$ be the currently assigned entity to $r$ and $t_y$, $t_d$ be the set of Yago and DBpedia classes for $e$.
16: If $t_y$, $t_d$ for $e$, do not contain either $top_y$ or $top_d$, send CHANGE message. Send $top_y$, $top_d$ as the classes for the entity that $r$ should update to; else send the NO-CHANGE message.
17: **end for**

The subclass relation between the DBpedia and Yago classes is obtained via the PARIS project [16].

If the alignment exists, then ignoring the lower score to either Yago or DBpedia, $\psi_3$ picks both the classes as the Yago and DBpedia assignments for the column header respectively. Otherwise, the class with the lower score is ignored, and the other one is selected as the column class. Once the column header is mapped to a class assignment, $\psi_3$ revisits each entity assignment in the column. All cell values (variable nodes) whose currently assigned entity $e$ include the top class(es) in their class set are sent a message of NO-CHANGE. Ones whose entity do not contain the top class in their class set are sent a CHANGE message. These variable nodes are also provided with the top class(es) as semantic preferences that their next entity assignment should try to fulfill. $\psi_3$ also sends the top score as a confidence score associated with the message.

$\psi_4$ – **Relations between pair of columns.** Algorithm 3 gives an overview of function that identifies relation between pair of columns in a table. The goal of factor node $\psi_4$ is to discover if a relation exists between a pair of columns, say *City* and *State*, and, if so, to use it as evidence to uncover any incorrect entity assignments in the columns' cells. At the end of *query and rank* phase, every row cell value has an initial entity assignment, $e$. The pair of column headers is also associated with a set of candidate relations. The initial assigned pair of entities in the two columns perform majority voting to select the best possible relation from the candidate relation set. Each pair of entities in every row between the two columns votes for a relation $rel$. It increments its score by 1 if $< e_{i,k} > < rel > < e_{j,k} >$ or $< e_{j,k} > < rel > < e_{i,k} >$ is true (*here i, j refer to two columns and k refers to entities from the k th row between the two columns*). Factor node $\psi_4$ queries Yago and DBpedia separately to check if the relations exists. The current $\psi_4$ implementation also maintains two separate sets of candidate relations (one for Yago and the other for DBpedia) ordered by votes.

Factor node $\psi_4$ also computes a value for *topScore* for both top Yago and DBpedia relations as the the number of votes divided by the number of cells in the column. If a score is below the current threshold, the relations are discarded and $\psi_4$ sends LOW-CONFIDENCE and NO-CHANGE messages to all of the row cell values in both columns and updates the relation between the columns to NO-ANNOTATION. If the scores are above threshold, the top Yago and DBpedia relations between the two columns are updated. $\psi_4$ then revisits the pair of entities from the columns to discover possible incorrect assignments. For every currently assigned entity $e$ in the two columns, $\psi_4$ checks if $e$ appears as a subject or object of the top relations (depending upon the relation direction; either Yago or DBpedia). If $e$ satisfies this constraint, a NO-CHANGE message is sent to the row cell and a CHANGE message is sent otherwise. The $\psi_4$ factor node also sends the relation information as characteristics the row cell should use for picking the next entity assignment and *topScore* as confidence score associated with the message.

**Updating entity annotations for row cell value variables.** Every row cell $r$ in the table receives messages from two types of factor nodes – column header

**Algorithm 3** $\psi_4$ – Relation between pair of columns

Let $YagoCandidateRels$ and $DBpCandidateRels$ be the set of Yago and DBpedia candidate relations between the columns $c_i$ and $c_j$.
Let $numberOfRows$ be the total number of rows in the table.
**for** k = 1 **to** $numberOfRows$ **do**
  Let $r_{i,k}$ and $r_{j,k}$ be the row values from columns $i$ and $j$ at row $k$.
  Let $e_{i,k}$, $e_{j,k}$ be the currently assigned entities for $r_{i,k}$ and $r_{j,k}$.
  **for all** $rel$ in $YagoCandidateRels$ **do**
    Majority Voting : If $< e_{i,k} > < rel > < e_{j,k} >$ or $< e_{j,k} > < rel > < e_{i,k} >$ is true, increment the score of $rel$ by 1
  **end for**
  Repeat the same majority voting process for candidate relations from $DbpCandidateRels$
**end for**
Sort the $YagoCandidateRels$ and $DbpCandidateRels$ sets in descending order. The relations at the top of the sets are the ones with maximum votes. Let $topRel_y$ and $topRel_d$ be the top Yago and DBpedia relations respectively. Let $topRelScore_y$ and $topRelScore_d$ be scores for $topRel_y$ and $topRel_d$, where $topScore = numberOfVotes \ / \ numberOfRows$
**if** $topRelScore_y$ and $topRelScore_d < relThreshold$ **then**
  Send message LOW-CONFIDENCE and NO-CHANGE to the row values in columns $c_i$ and $c_j$.
  Update relation annotation to NO-ANNOTATION.
**end if**
**if** $topRelScore_y >= relThreshold$ **then**
  Update the top Yago relation between the columns $c_i$ and $c_j$ to $topRel_y$.
**end if**
Similarly update the top DBpedia relation between columns $c_i$ and $c_j$.
**for** k = 1 **to** $numberOfRows$ **do**
  Let $e_{i,k}$, Let $e_{j,k}$ be the currently assigned entities for $r_{i,k}$ and $r_{j,k}$.
  If $< e_{i,k} > < topRel > < e_{j,k} >$ and $< e_{j,k} > < topRel > < e_{i,k} >$ is false, send theCHANGE message to the row value, whose entity violates the relation. Send top relation as the relation row variable entity assignment should participate in. ($topRel$ can be either $topRel_y$ or $topRel_d$)
  Else, send theNO-CHANGE message to both $r_{i,k}$ and $r_{j,k}$.
**end for**

factor node ($\psi_3$) and relation factor node ($\psi_4$). While $r$ will receive only one message from the column header (since $r$ belongs to only one column), it might receive multiple messages from relation factor nodes if its column is related to several columns in the table. For example, the column *City* is associated with columns *State*, *Mayor* and *Population*. The result is that $r$ can receive conflicting messages – some factor nodes might send a CHANGE message, while others a NO-CHANGE message.

If all of the messages received by $r$ are NO-CHANGE, $r$ does not update. If all messages received by $r$ are CHANGE, it decides to update its assignment. In the case of conflicting messages, $r$ uses the confidence score sent by each factor node along with the message to compute the average score associated with the CHANGE messages and compares it against the average score associated with the NO-CHANGE messages. If the average CHANGE message score is higher, $r$ updates its current assignment, otherwise it does not.

When $r$ chooses to update its current assignment, it picks a new assignment based on the semantic preferences sent by the factor nodes. For example, a row cell value in the first column of the table in Figure 2 might receive messages to update to an entity which will have a *rdf:type City* and is the subject of relations *isPartOf* and *hasMayor*. The row cell value $r$ iterates through its ranked list of candidate entity set and picks the next best entity satisfying all the semantic preferences specified in the message. In cases where $r$ cannot find an entity that satisfies them all, it orders them based on the confidence scores associated with the respective messages and attempts to pick an entity assignment that satisfies the highest rank combination. For example, if there were three preferences, ranked *1*, *2* and *3*, $r$ will first attempt to find an entity that satisfies*[1,2,3]* followed by *[1,2]* ; *[1,3]*; *[2,3]*; *[1]*; *[2]* and so on. If $r$ is unable to find an entity that satisfies any preferences, then it updates its current assignment to NO-ANNOTATION.

An exception to this process occurs when the candidate entities for the cell all have low confidence (i.e., below the threshold ($index\_threshold$)). This is typically the case if the entity is absent from the knowledge base. In such cases, the algorithm maps the row cell value to NO-ANNOTATION rather than linking to any candidate. If the column header is mapped to NO-ANNOTATION, the row cell values retain the top ranked entity assignment as suggested by the entity ranker.

**Halting condition.** Once the row cell values have updated, they send their new assignments to the factor nodes and the entire process repeats. Ideally, the process should be repeated until best possible assignments are achieved; i.e., repeat until no variable node receives a CHANGE message or none of the variable nodes select a new assignment. Practically, this a hard convergence metric and it is often not achieved. In our current implementation, the *Semantic Message Passing* algorithm lets this cycle repeat for five iterations. After five iterations, the algorithm checks the number of variables that have received a CHANGE message. If the number of variables is lower than the threshold required to update the column header or relation annotation, the process is stopped, else the pro-

cess continues until convergence or the tenth iteration has been completed. The assignments at the end of the final iteration are chosen as final values.

## 3 Evaluation

We begin by describing the experimental setup and follow by presenting our evaluation and analysis for column header, cell value and relation annotations, and performance of the graphical model in terms of convergence and running times.

**Experimental Setup.** We used tables from four different sets in our evaluation (see Figure 6). The original table sets, obtained from [10], include ground truth annotations in which column headers are mapped to Yago classes, relations between columns to Yago properties and row cell values linked to Yago entities. However, we could not use these assessments for column headers and relations, since our system uses data from both DBpedia and Yago, and thus developed our own gold standard. We ran our factor nodes $\psi_3$ and $\psi_4$ at low threshold (5%) to generate candidate classes and relation between table columns. We presented these candidates along with raw tables to human annotators, who marked each as *vital*, *okay* or *incorrect* (as in [20]). For example, annotators could mark the label *City* as *vital*, *PopulatedPlace* as *okay* and *Person* as incorrect for the first column of the table in Figure 2. Thus each column can have multiple *vital* and *okay* class labels as per annotator judgment.

| *Dataset* | *Col & Rel* | *Cell Value* | Avg.[Col,Row] |
|---|---|---|---|
| *Web_Manual* | 150 | 371 | [2,36] |
| *Web_Relation* | 28 | – | [4,67] |
| *Wiki_Manual* | 25 | 39 | [4,35] |
| *Wiki_Links* | – | 80 | [3,16] |

Fig. 6: Number of tables and the average number of columns (col) and rows in the sets used for column header, cell value and relation annotation. Average is over the number of tables used in cell value annotation.

For the evaluation that follows, we used the framework with the following values: for every cell value we chose the top 25 candidate entities from the entity ranker; the column header top score threshold used by $\psi_3$ and the relation top score threshold used by $\psi_4$ were set to 0.5; *index_threshold* used by a row cell value to determine low confidence entities was set to 10. The number of joint inference model iterations is as described in the previous section.

**Evaluating column header annotations.** We generated a ranked list of at most top ten class annotations for every column, with NO-ANNOTATION as the single value if no appropriate class was found. We compared the set of generated classes to those obtained from annotators, computing precision and recall for each $k$ between 1 to 10. For precision, we assign a score of 1 for every *vital* class and 0.5 for every *okay* class identified by the framework. For recall, we assign 1 for every *vital* and *okay* class identified. This evaluation scheme is similar to the one described in [20].

Figure 7(a) shows the annotation results for class labels at rank 1 across three different sets. We observed that a single column had more than one label marked *vital* or *okay* by annotators and the recall increasing with $k$. The lower

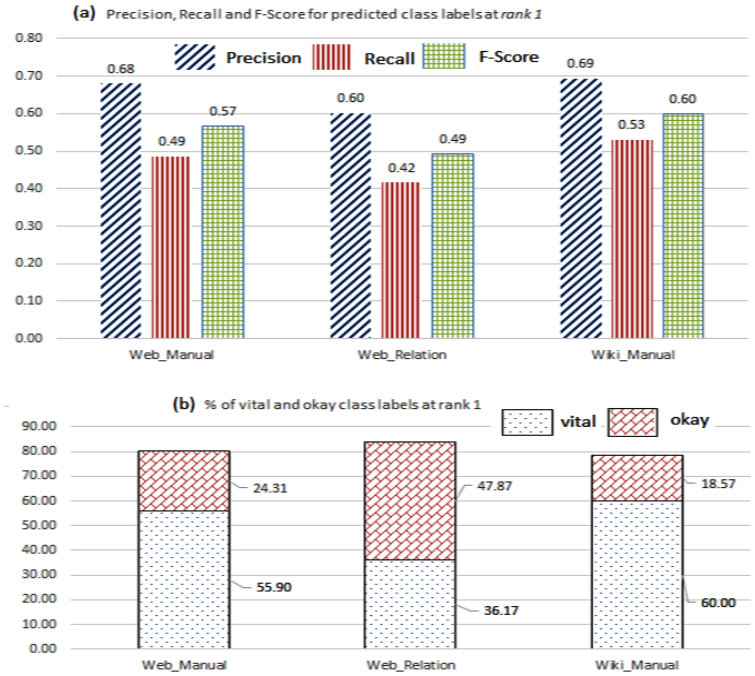


Fig. 7: (a) Precision, Recall and F-scores for column header annotations. (b) % of *vital* and *okay* class labels at rank 1.

recall can be explained by the combination of having multiple labels for each column and only one label retrieved at *rank 1*. Due to space constraint, we do not show results for precision and recall for values of $k$ between two and ten.

We also computed how often the label predicted at rank 1 was either *vital* or *okay*, as shown in Figure 7(b). The high percentage of a combination of *vital* and *okay* labels at rank 1 for all three sets (*79% or greater*) indicates that the top ranked labels are relevant. The results for the three sets are for the classes from the DBpedia ontology. We note that our F–scores for *Web_Manual (0.57)* and *Wiki_Manual (0.60)* are better than the previously reported scores of 0.43 and 0.56 in [10]. We fare slightly poorer for both datasets against scores of 0.65 and 0.67 as reported in [20]. However, we note that their evaluation is over the annotations of the original ground truth in which every column header had only one, or sometimes two, correct classes. This leads to a better recall at rank 1, as compared to multiple correct classes, as in our case, with a combination of *vital* and *okay* labels. We also note that the task of the system in [20] is predicting classes for column headers and relation between "primary column" (e.g., a key) and other columns in the table independently. Our framework, in contrast, attempts to jointly map column header, cell values, relation between columns to appropriate assignments which in certain cases can lead to incorrect assignments.

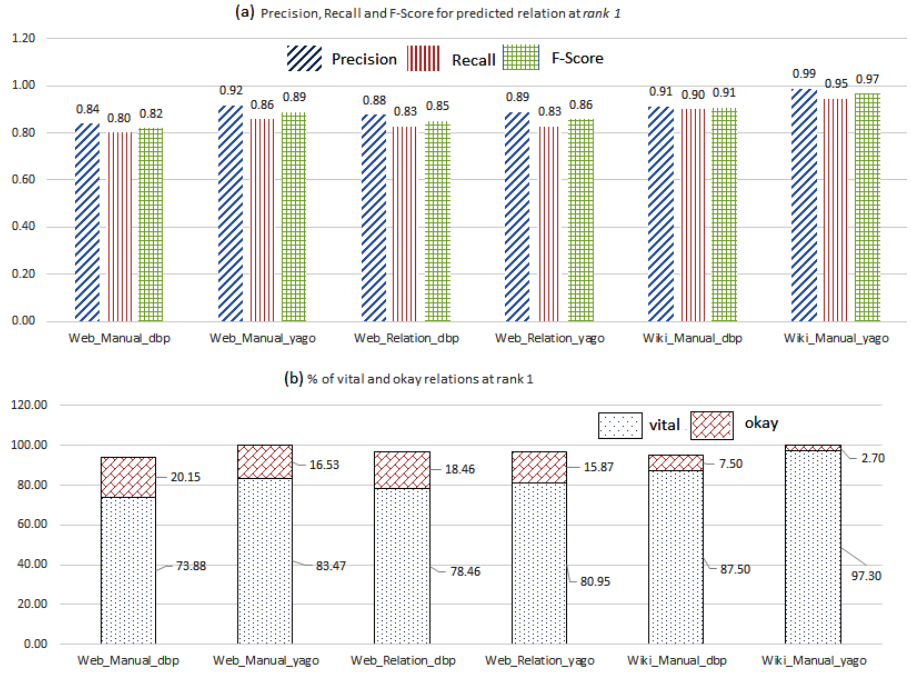


Fig. 8: (a) Precision, Recall and F-scores for relation annotations. (b) % of *vital* and *okay* relation labels at rank 1, as predicted by the framework. Dataset names followed by a *_dbp* are results for DBpedia relations; whereas the ones followed by *_yago* are for Yago relations.

**Evaluating relation annotations.** We generated and ranked the best ten relations (as RDF properties) both from DBpedia and Yago. We compared the set of relations generated to those obtained from the annotators, computing precision and recall for $k$ from 1 to 10 as described in the previous section. Figure 8(a) shows the results across four different sets for both Yago and DBpedia relations. While it is plausible for a column header to have multiple *vital* and *okay* classes, the same may not hold true for relation between columns. We observed in our annotations that for every pair of columns, the set of *vital* and *okay* relations was smaller.

We explored possible relations between all pairs of columns, even though tables typically represent a small number of relations. Typically a table's primary (e.g., key) column participates in binary relations with many or most of the other columns in the table. Thus our system generates a fair number of NO-ANNOTATION labels. We believe both these reasons explain higher values for precision and recall. We also compute the percentage of *vital* and *okay* relations predicted by the framework at rank 1 (see Figure 8 (b)). Analogous to precision and recall results, a high percentage of labels were *vital* and *okay* indicating that our framework is generating relevant labels. Our F–scores for relation annotations for the datasets *Web_Manual_yago (0.89)*, *Web_Relation_yago(0.86)* and

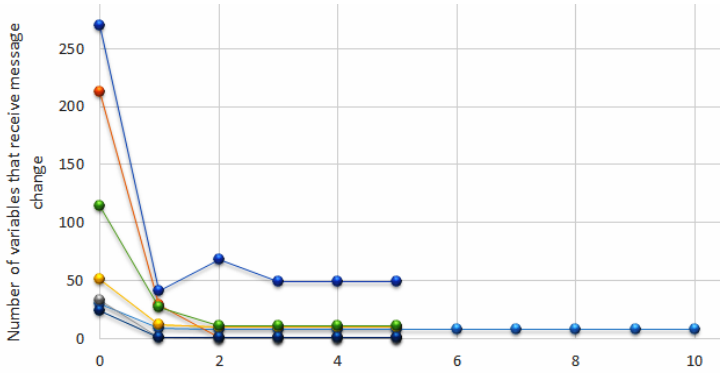


Fig. 9: X is the iteration number and Y the number of variables that received CHANGE message in that iteration. The value at x=0 is the number of cells in the table.

*Wiki_Manual_yago(0.97)* fare better as compared to the scores of 0.51, 0.63, 0.68 reported in [10].

**Evaluating Cell value annotations.** We compared the entity links generated by our framework to those obtained as ground truth from the original dataset [10]. Our framework linked the table cell value to an entity from DBpedia wherever possible, else it linked to NO-ANNOTATION. If our predicted entity link matched the ground truth, we considered it as a correct prediction, else incorrect. We obtained an accuracy of 75.89% over the *Wiki_Links* dataset; 67.42% over the *Wiki_Manual* dataset and 63.07% over the *Web_Manual* dataset. Lower accuracy for entity linking is likely due to the lack of relevant data in Yago and DBpedia. Although our framework might have discovered the correct assignments for column header and relation, if the entity did not have the class and relation information present on DBpedia or Yago, our framework will fail to find it.

For example, even if we discover the Yago class *YagoGeoEntity* which links all places, the DBpedia *Berlin* entity does not have that class, thus can lead to an incorrect assignment. Lower accuracy may also stem from the size of the candidate entity set. We restricted the size of the candidate entity set to 25; thus it is possible that the correct assignment could be outside this set. We also note certain discrepancy in the ground truth annotations. We discovered cases where we were able to discover a correct entity annotation, whereas the ground truth said it was NO-ANNOTATION, which led to counting our annotation as incorrect. We are working on improving the ground truth annotation and we have not incorporated the results from the updated ground truth.

**Graph Convergence.** Figure 9 gives insight into how quickly the graph converges, showing the number of variable nodes that receive a CHANGE message at the end of every iteration (for eight tables). For most tables, the number of variable nodes that receive a CHANGE message stabilize after the first iteration. We had few cases, where the variable count fluctuated, i.e., increasing and decreasing as iterations increased. We also noticed cases where the variable count does

not go to zero. Some number of "stubborn" variables keep receiving a CHANGE message at the end of every iteration, but cannot find a new value. However, we noticed that the number of stubborn variables are less as compared to the original number of variables in the table. We present results for eight tables for visual purposes; the results are representative of rest of the tables in the dataset. The average time required for the inference model across all tables was 3.4 seconds.

**Entity Ranker.** The *entity ranker* uses a classifier that produces likelihoods that strings should be linked to entities. The training and test datasets were generated using the ground truth for entity annotations from *Wiki_Links* set. For every string mention in the table, we queried Wikitology to get candidate entities and then computed feature values for the string similarity and popularity metrics for each mention/entity pair. A class label of 1 was assigned if the candidate entity was the correct assignment (available via ground truth in the dataset) and a 0 otherwise. The training set included 600 instances, evenly split between positive and negative instances. The test set included in all 681 instances with 331 positive and 350 negative instances. Out of the 681 instances, the model was able to correctly classify 619 instances with an accuracy of *90.9 %*. The precision, recall and F-score are presented in Figure 10.

| *Class* | *Precision* | *Recall* | *F-Score* |
|---|---|---|---|
| *0* | 0.959 | 0.849 | 0.901 |
| *1* | 0.871 | 0.966 | 0.916 |

Fig. 10: Precision, recall and F-score for the Naive Bayes model.

## 4 Related Work

Our work is related to two threads of research, one focused on pragmatically generating RDF from databases, spreadsheets and CSV files and a more recent one that addresses inferring the implicit semantics of tables. Several systems have been implemented to generate semantic web data from databases [15, 19, 13], spreadsheets [6, 9] and CSV files [3]. All are manual or only partially automated and none has focused on automatically generating *linked* RDF data for the entire table. In the domain of open government data, for example, [3] presents techniques to convert raw data (CSV, spreadsheets) to RDF but the results do not use existing classes or properties for column headers, nor does it link cell values to entities from the LOD cloud. Generating richer, enhanced mappings requires a manually constructed configuration file.

Early work in table understanding focused on extracting tables from documents and web pages [7, 5] with more recent research attempting to understand their semantics. Wang et al. [21] began by identifying a single 'entity column' in a table and, based on its values and rest of the column headers, associate a concept from the Probase knowledge base with the table. Their work does not attempt to link the table cell values or identify relations between columns. Ventis et al. [20] 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. Their work also does not attempt to link the table cell values. Limaye et al. [10] 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 more semantics, including relations between column headers and between row entities. Current table interpretation systems rely on semantically poor and possibly noisy knowledge-bases and do not attempt to produce a complete interpretation of a table. None generate high quality linked data from the inferred meaning or can interpret columns with numeric values and use the results as evidence in table interpretation, a task essential for many domains.

## 5 Conclusions

Generating an explicit representation of the meaning implicit in tabular data will support automatic integration and more accurate search. In this paper, we presented an implementation of our graphical model which infers a table's meaning relative to a knowledge base of general and domain-specific knowledge. We described a novel *Semantic Message Passing* algorithm which avoids computing potentially huge joint-probability distribution tables normally required in such graphical models.

A thorough evaluation showed promising results, but leaves room for improvement. We believe our extensible and domain independent framework can address the existing challenges in converting tabular data to RDF or high quality linked data. In the future, we will work on designing a cooperative environment in which a person-in-the-loop identifies bad system choices for column classes and relations, cell value entities and optionally suggests better ones from the alternate candidates.

**Acknowledgement.** This work was supported by AFOSR award FA9550-08-1-0265 NSF awards 0326460 and 0910838 and a gift from Microsoft Research.

## References

1. Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., Hellmann, S.: Dbpedia - a crystallization point for the web of data. Journal of Web Semantics 7(3), 154–165 (2009)
2. Cafarella, M.J., Halevy, A.Y., Wang, Z.D., Wu, E., Zhang, Y.: Webtables: exploring the power of tables on the web. PVLDB 1(1), 538–549 (2008)
3. Ding, L., DiFranzo, D., Graves, A., Michaelis, J.R., Li, X., McGuinness, D.L., Hendler, J.A.: TWC data-gov corpus: incrementally generating linked government data from data.gov. In: Proc 19th WWW. pp. 1383–1386. ACM (2010)
4. Dredze, M., McNamee, P., Rao, D., Gerber, A., Finin, T.: Entity disambiguation for knowledge base population. In: COLING. pp. 277–285 (2010)
5. Embley, D.W., Lopresti, D.P., Nagy, G.: Notes on contemporary table recognition. In: Document Analysis Systems. pp. 164–175 (2006)
6. Han, L., Finin, T., Parr, C., Sachs, J., Joshi, A.: RDF123: from Spreadsheets to RDF. In: Proc. 7th Int. Semantic Web Conf. Springer (October 2008)

7. Hurst, M.: Towards a theory of tables. IJDAR 8(2-3), 123–131 (2006)
8. Koller, D., Friedman, N.: Probabilistic Graphical Models: Principles and Techniques. MIT Press (2009)
9. Langegger, A., Wob, W.: Xlwrap - querying and integrating arbitrary spreadsheets with SPARQL. In: Proc. 8th Int. Semantic Web Conf. (October 2009)
10. Limaye, G., Sarawagi, S., Chakrabarti, S.: Annotating and searching web tables using entities, types and relationships. In: Proc. 36th VLDB (2010)
11. Mulwad, V., Finin, T., Joshi, A.: A Domain Independent Framework for Extracting Linked Semantic Data from Tables. In: Search Computing - Broadening Web Search, pp. 16–33. Springer (July 2012), lNCS volume 7538
12. Mulwad, V., Finin, T., Syed, Z., Joshi, A.: Using linked data to interpret tables. In: Proc. 1st Int. Workshop on Consuming Linked Data. Shanghai (2010)
13. Polfliet, S., Ichise, R.: Automated mapping generation for converting databases into linked data. In: Proc. 9th Int. Semantic Web Conf. (Nov 2010)
14. Puranik, N.: A Specialist Approach for Classification of Column Data. Master's thesis, University of Maryland, Baltimore County (August 2012)
15. Sahoo, S.S., Halb, W., Hellmann, S., Idehen, K., Thibodeau Jr, T., Auer, S., Sequeda, J., Ezzat, A.: A survey of current approaches for mapping of relational databases to rdf. Tech. rep., W3C (2009)
16. Suchanek, F.M., Abiteboul, S., Senellart, P.: PARIS: Probabilistic Alignment of Relations, Instances, and Schema. PVLDB 5(3), 157–168 (2011)
17. Suchanek, F.M., Kasneci, G., Weikum, G.: Yago: A Core of Semantic Knowledge. In: 16th Int. World Wide Web Conf. ACM Press, New York (2007)
18. Syed, Z., Finin, T.: Creating and Exploiting a Hybrid Knowledge Base for Linked Data. Springer (April 2011)
19. Vavliakis, K.N., Grollios, T.K., Mitkas, P.A.: RDOTE- transforming relational databases into semantic web data. In: 9th Int. Semantic Web Conf. (2010)
20. Venetis, P., Halevy, A., Madhavan, J., Pasca, M., Shen, W., Wu, F., Miao, G., Wu, C.: Recovering semantics of tables on the web. In: Proc. 37th VLDB (2011)
21. Wang, J., Shao, B., Wang, H., Zhu, K.Q.: Understanding tables on the web. Tech. rep., Microsoft Research Asia (2011)
22. Zagari, R., Bianchi-Porro, G., Fiocca, R., Gasbarrini, G., Roda, E., Bazzoli, F.: Comparison of 1 and 2 weeks of omeprazole, amoxicillin and clarithromycin treatment for helicobacter pylori eradication: the hyper study. Gut 56(4), 475 (2007)