Ontology Guided Autonomous Label Assignment in Wrapper Induced Tables with Missing Column Names*

Mohammad Shafkat Amin Hasan Jamil
Department of Computer Science, Wayne State University, USA
shafkat@wayne.edu, jamil@cs.wayne.edu

Abstract

Formulating and executing queries over distributed, autonomous and heterogeneous resources is an important research area. The advent of the Internet and the Web and their inherent ubiquity have brought forth opportunities to query these information sources in an automated and independent manner. In the domain of information extraction, automatic wrapper generation has been well studied but the efficacy of the current wrappers are limited by the fact that automatic annotation of column names to the extracted tabular data is yet to be perfected. In this paper, we propose a novel annotation system that can assign meaningful column names to the extracted tables for subsequent queries. We enhance our prototype wrapper system FastWrap with this annotator to support fast and autonomous on-the-fly data integration and ad hoc declarative querying.

1 Introduction

In recent years, there has been a flurry of research in the area of web data extraction and wrapper generation. Except a few, most research focused on producing end results for human consumption, and thereby did not address the issue of missing column names. The issue of missing column names arise for two primary reasons. First, typical Web pages often omit labels, which are understood from the context by a human. Second, even if a column name is present, user is forced to use the labels chosen by the Web content providers, which may not be the most appropriate or most descriptive ones.

For example, if we take a look at the response produced by a query submitted at www.dell.com as shown in figure 1, we can see that none of the attributes in the resulting records contain a column name. Thus to be able to query the site considering it as a relational data repository, not only the data record will be needed to be extracted but also all the constituent attributes will need to be named.

Figure 1. Example extraction scenario

In applications such as comparison shopping where large number of “arbitrary” web sites are searched and information extracted, manual name assignment is not feasible, and an automated procedure is preferred so that results of a query can be used in a follow-up query or can be composed with other queries. Examples of such comparison shopping engines include expedia.com, shopzilla.com, amazon.com, buy.com, and kayak.com. These engines are designed to access specific set of hidden web sites and consolidate information using one single web form so that users need not browse the individual sites themselves manually. While these engines are efficient and have their advantages, there is a need to integrate hidden web databases dynamically and on demand. This is because all needs cannot be specified ahead of time, new databases emerge every day and needs to be integrated with the existing engines and in the event of changes in the databases, the search engine needs to adapt autonomously to the change. The motivation of an automated tool comes from the fact that, these comparative shopping sites are neither capable of handling newer sites, unless they are explicitly designed to integrate specific sites, nor are they robust against changes.

*Research supported in part by National Science Foundation grants CNS 0521454 and IIS 0612203.
in sites that they do automate extraction from. Moreover, the user requirement is not limited to the any specific subset of web sites. A user may want to submit a notebook configuration and price related query on all the available vendor sites returned from a Google search. This needs explicit handling of curating information from thousands of web sites and goal driven comparison of the obtained result. This query can neither be performed manually, nor can it be served by any available comparative shopping site because this requires on-the-fly generation of automatic, accurate and efficient wrapper, which is not handicapped by the autonomy of the sites.

For example, for the user query to find the cheapest laptop with Intel pentium IV processor from all the vendor sites returned from the Google search with the term shopping online laptop, the user has to take into account a dynamic number of vendor pages with possible multiple potential structures and semantics to analyze. Thus, with the automated information integration in the horizon, there is no limit to what level of flexibility a user can achieve. In Data Extraction and Label Assignment (DeLA) [9], the main idea of the labeling process is that form elements will probably re-appear in the corresponding fields of the data objects. To assign labels to the columns of the data table containing the extracted data objects, i.e. to understand the meaning of the data attributes, the following four heuristics were exploited in DeLA:

**Heuristic 1** Match form element labels to data attributes

The search form of a web site through which users submit their queries provides a sketch of the underlying relational database of the web site. If we make the assumption that the web site designers try their best to answer user queries with the most relevant data, then keyword queries submitted through one specific form element will reappear in the corresponding attribute values of the data objects. Therefore, for each form element with its keyword queries, if the keywords mostly appear in one specific column of the data table, then we can assign the label of that form element to the column.

**Heuristic 2** Search for voluntary labels in table headers

The HTML specification defines some tags such as `<TH>` and `<THEAD>` for page designers to voluntarily list the heading for the columns of their HTML tables. Moreover, those labels are usually placed nearby the data objects. Therefore, the HTML code near (usually on the top of) the contained data objects is examined for possible voluntary labels. We need to augment header using an ontology i.e representative terms of a domain. In many situations the table header alone is not enough to describe the semantics of that table. Another approach is then used, namely the table header is used to extract the corresponding concept from the domain ontology and search for that concept and all related synonyms in the text.

**Heuristic 3** Search for voluntary labels encoded together with data attributes

Some web sites encode the labels of data attributes together with the attribute values. Therefore, for each column of the data table we try to find the maximal-prefix and maximal-suffix shared by all cells of the column and assign the meaningful prefix to that column and the meaningful suffix to the column next to that column as the labels.

**Heuristic 4** Label data attributes in conventional formats

Some data have a conventional format, e.g. a date is usually organized as dd-mm-yy, dd/mm/yy, email usually has the symbol @, price usually has the symbol $, etc. Thus, such information is used to recognize the corresponding data attributes. Note that the form elements and the data attributes do not need to be perfectly matched. Therefore, the label assigner may not be able to assign meaningful labels to all of the data attributes. DeLA also allow users to add a label to unassigned attributes and to modify the assigned labels.

As discussed above, DeLA demonstrated the feasibility of heuristic-based label assignment and the effectiveness of the employed heuristics, which set the stage for more fully automatic data annotation of web sites. Since web pages are designed to be presented on a browser to a human user, usually values and labels are visually close to each other. Therefore, first Labeller [6] computes the coordinates of the bounding boxes of every data-value and every label in a given sample page. Then it tries to find the optimal association label/data-value by analyzing their spatial relationships. Arlotta et al [2] also have developed several heuristics to establish the correct associations:

1. values and labels are close to each other
2. usually a label is vertically, horizontally, or centrally aligned to its associated values
3. labels are usually placed either to the left or above values
4. it is not allowed that either a label or a value is between another value and its label

The wrappers produced by [5], on the other hand, need a post-processing phase to annotate the extracted attributes with more semantic labels, which are initially anonymous. Visual Perception-based Extraction of Records (ViPER) mainly developed two heuristics: intercolumn label assignment heuristics and inner-column label assignment heuristics to assign column label to data items.

### 2 Wrapper Generation System FastWrap

Wrappers are specialized program routines that automatically extract data from Internet web sites and convert the
information into a structured format. Published information in many web pages in the WWW is based on databases running in the background. When compiling this data into HTML documents, the structure of the underlying databases is completely lost. Wrappers try to reverse this process by restoring this information to a structured format. It is only intuitive to address this extraction process by means of obtaining candidate patterns from the input page. FastWrap [1] is such a wrapper generation system that extracts tabular data from HTML pages by searching for regular patterns. In FastWrap, we employ suffix tree based technique to obtain records which we term tabular data. After the pattern has been extracted, it is refined using KMP-prefix (Knuth-Morris-Pratt) algorithm and then it is converted into a regular expression which is subsequently used to extract the record level data items. KMP-prefix algorithm is applied to prevent extraction of patterns that may have part of its prefix appended in the end.

FastWrap
Input: HTMLPage Page.
Output: TableData T.
begin
1 str ← convert HTML Page to symbol list
2 st ← build suffix tree for str
3 lrp ← get the longest repeated/super-maximal pattern
4 r/l ← get modified repeated patterns after applying KMP
5 Ci = Ci − Π − Σ;
6 For each p in pattern
7 apply circular modified alignment to p
8 T = Extract target information from Page using p
9 return T
end

An HTML page contains tags as well as texts. Among all the tags that an input HTML page may contain, some of them are not of any interest for the whole wrapper generation process. A non-exhaustive list of tags that are unimportant for the extraction process is given below.

<\br>, <\nobr>, <\wbr>, <\script>, <\style>, <\b>, <\meta>, <\noscript>, <\input> etc.

So these tags and any information enclosed within these tags are discarded from the input HTML page. Tags like <\script>, <\style>, and <\meta> are required only for the processing purposes, and thus are also considered unimportant for extraction. Subsequently, the input HTML page is converted into a list of symbols, where every tag and the ending tag (if exists) associated with it is given a unique symbol representation. We use hash tables to map between unique HTML tag items and their corresponding symbols.

1Many other unimportant tags have been identified in the literature from empirical experiments which can be included in this set to further improve the extraction process.

subsequently construct a suffix tree from the symbolized HTML input page for pattern extraction. Next we obtain the largest repeated pattern from the suffix tree.

Given a string T with |T| = n, where n > 0, the longest repeated substring problem is to identify and locate the longest substring x, occurring at two or more distinct, possibly overlapping, positions in T. It is an easy application of the suffix tree to find it in O(|T|) time, where |T| represents the length of T, i.e., constructing and extracting longest repeated substring in suffix trees is linear time solution [8].

The internal nodes of a suffix tree represent common prefixes of the suffixes of the input string. It is thus evident that the longest substring represented by an internal node in the tree is the longest repeated substring of the input.

The longest repeated substring extracted from the suffix tree may contain overlapped data which, from the context of web wrapper generation, may pose to be spurious. For example, for the string abcdabcdabc, the longest repeated pattern returned by the suffix tree would be abdabc, which clearly is not our pattern of interest. This is due to the fact that the pattern abdabc has the suffix abc which is also a prefix of the pattern. This may denote that the suffix abc is a member of the subsequent record of the input. Thus the second abc beginning at position 5 in the input string is basically a member of the second "abcd" beginning at position 5. To refine this pattern we employ KMP prefix computation function [4].

The prefix computation is formalized as follows: given a pattern P[1...m], the prefix function for the pattern P is the function π : {1,2,3,...,m} → {0,1,...,m−1} such that π(q) = max{k : k ≤ q and Pk ⊆ Pq}. That is π(q) is the length of the longest prefix of P that is a proper suffix of Pq and Pq is the prefix of P of length q. Thus we calculate π(q) for all q and extract π(m), where m is the length of the pattern. If L is the length function, then L(P) = m. So our pattern of interest would be, P_{L(P)−π(L(P))} or P_{m−π(m)}.

The pattern extracted from the previous step is converted into a regular expression next. Whenever a text item is encountered in the pattern, it is replaced by the regular expression ^<\^>|∗\^>, where "^" in this context defines that anything but the characters < and > can appear in the matching text. Thus any non-tag item is considered as text. All the tags are concatenated with the string ^<\^>|∗\^> to account for the attributes and the parameters associated with a tag. Thus the regular expression for a Table tag would be like ^<Table[^|∗[^>|∗]|^>\^>|∗\^>|∗\^>|∗>\^>|∗>\^>|∗>|∗>\^>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗>|∗|
more than one candidate pattern reflecting user expectation, the choice of either maximal repeat or super-maximal repeat seem logical. This myriad of patterns, once extracted can be ranked according to an objective function that reflects syntactic and semantic comprehension of the domain in question. For the simplest use case, a function of pattern size and number of repeats can be used as parameters for the objective function. The extraction of all maximal repeats and super maximal repeats can be executed in linear from a constructed suffix tree [7].

The patterns thus obtained must undergo the same purging and filtering process before they are ranked to be used for practical purposes. In order to facilitate extraction of nested fields and missing attributes, the top-k patterns extracted are then compared using edit distance with the highest ranked pattern to test whether the inclusion of this pattern would improve overall performance. Let \( P_0 \) be the highest ranked pattern and \( P_i \) be any other Pattern. Let \( ED = EditDistance(P_0, P_i) \). For all patterns with \( ED \leq T_0 \), where \( T_0 \) is a user given threshold, participating data items are extracted and all the data items are subsequently aligned using standard multiple sequence alignment technique.

### 3 A Procedure for Column Name Annotation

While FastWrap can extract tables efficiently, it is currently unable to assign names to columns without a name. The goal of this paper to propose a procedure to do so and enhance FastWrap. But the technique we propose is general enough to be applicable in other wrapper systems as well. As discussed, wrapper generation process from web data sources entails extraction of tabular structure from the input HTML page and its conversion into a relational structure for subsequent query management. This process introduces the challenge of annotating extracted columns from the input HTML text so that the assigned annotations can be used to treat the result as a relational table. The process itself inherently introduces newer complexities regarding domain dependent labeling. In this paper, we propose a novel algorithm to address these issues by analyzing both syntactic and semantic information embedded in the data. In order to impute putative representative column names to be assigned in each of the data fields extracted from the input HTML table, we maintain a hierarchy of Ontological knowledge-base pertaining to the semantic and structural information which is subsequently applied to the column data vector to extrapolate its membership column ID from a list of input column names. Data table, once extracted is subjected to a dual scanning mechanism i.e. horizontal and vertical. The vertical scan is performed for each column, where we strive to extract the structural commonality that may exist among the data pertaining to the same column. The structural information is expressed in regular expression in terms of numeric characters, special characters, and string of alphabets.

In order to further restrict the scope of membership, any alphanumeric string that appears in each and every row of the column under consideration, is retained in the structural information. For Example, In flybase (http://flybase.org) all the annotation IDs contain the prefix “CG” followed by a set of numeric characters. So in this case the structural information will be CG[n]+ instead of [s]+[n]+, where \( s \) and \( n \) stand for alphabet and numeric characters respectively. Similarly, Flybase IDs for all the genes in the database contain “FBgn” as their prefix. Thus retaining this information in the structural pattern of the column will lead to a pruned search space. In case of events where the structural pattern information look up leads to multiple hits in the ontology, we maintain a pointer to the semantic keyword lookup system, which is a myriad of knowledge annotation tools (e.g. Genia Tagger tool (http://www-tsujii.is.s.u-tokyo.ac.jp/GENIA/tagger/), Stanford parser http://nlp.stanford.edu/software/lex-parser.shtml that can be used to annotate DNA, RNA, Protein, Gene names and so on).

The ontology is designed to have a flexible and extendible structure, where a previously unseen structural and semantic information can be easily incorporated in the hierarchical repository. The horizontal scan is performed to mitigate contention between two or more columns that have the probability of being assigned the same column ID. As a first step of the processing we initially iteratively process data pertaining to a single column to identify hidden structural patterns associated with it. We first extract the largest common prefix and largest common suffix pertaining to all values in the column vector. If so, we retain those values since they are likely to provide semantic information. Next, we split each and every entry in the column vector in terms of constituent parts of structural components and create a generic regular expression corresponding to that entry. However, it is likely that not all of the entries in a column vector will adhere to the same structural pattern. Hence we combine all the entries by creating a global regular expression that can satisfy all the elements in the column vector. This is done by vertically aligning the structures using multiple sequence alignment technique.

For example, an address column may contain the following values: Michigan 48201, New York, Ohio. Once we construct for each of the entries a structural pattern, we will end up with the following expressions respectively:

\[
[a - zA - Z\backslash s] + [0 - 9\backslash s]+, [a - zA - Z\backslash s]+,
[0 - 9\backslash s]+ [a - zA - Z\backslash s]+, [a - zA - Z\backslash s]+
\]

If we let \( \beta = [a - zA - Z\backslash s]+ \) and \( \mu = [0 - 9\backslash s]+ \), we can rewrite the expressions as \( \beta \mu, \beta, \mu \beta, \beta \). Now we need to combine these expressions to obtain a global regular ex-
pression. We employ clustalw multiple sequence alignment technique first to align the extracted patterns. In this case the alignment looks as follows:

1 2 3
- β µ
- β -
µ β -
- β -

Thus in order to generate a generic regular expression we obtain the following pattern: \[|β|e|^β|e]|, where e denotes empty string. This is due to the fact that if '-' appears in any of the vertical character entries in the alignment, it implicitly denotes that any character other that '-' that appears in that vertical position (e.g. in the above example position 1, 2 or 3) will be an optional character since it does not appear in all the rows of the column vector. The algorithm for syntactic data extraction can now be written as follows.

Procedure ExtractStructure
Input: ColumnVector C.
Output: Labeled ColumnVector R.
begin
1 Structure S[1…length(C)]
2 \(Σ = \text{LongestCommonPrefix}(C)\)
3 \(Π = \text{LongestCommonSuffix}(C)\)
4 For each element \(i ∈ C\)
5 \(C_i = C_i − Π − Σ\)
6 For each element \(i ∈ C\)
7 For each element \(j ∈ \text{RegexPatternin}\)
8 \(S = \text{Substitute}(M, C_i, p_j)\)
9 For each element \(k ∈ S\)
10 \(S_k = \text{concat}(Π, S_k, Σ)\)
11 \(R = \text{GenerateGenericRegularExpression}(\text{ClustalW}(S))\)
12 Return R
end

Once we have obtained the generic regular expression from the column we look up the Ontology that allows traversing its hierarchy based on the query pattern to reduce the search space in the semantic matcher domain. If the structure is absent in the Ontology, the user can incorporate the structure in the ontology. The actual level in the ontology that is chosen for the storage of that entry is obtained by choosing the node in the ontology for which the edit distance of the input pattern and the pattern associated with the node is minimum. Each Node contains a list of plausible keywords that is associated with the structure. Since the whole wrapper extraction process is driven as a response to a user query where a user submits a list of column names to be extracted, we employ OntoMatch [3] to filter the list of names that can be assigned to a specific column. Although the user expected column names may differ significantly from the actual annotation, OntoMatch can infer the semantic and syntactic relationship between the user expectation and the system assigned annotation.

In figure 2, we can see the part of the ontology that defines the concept “Address”. At the highest level of the concept is the generic root that contains all the variants of the concept itself. The concept can be formed by numerous sub-concepts as depicted by the children nodes in the ontology or it can be defined in terms of a terminal structure as well (depicted by the rectangle labeled “Address Structure”). This structure gets appended and modified as more and more web sites are visited. OntoMatch is a hybrid matcher, meaning that, given a set of patterns \(M = m_1, m_2,…, m_n\), for every pair of terms \((t_1, t_2)\), OntoMatch first selects \(M' ⊆ M\) and then applies matchers successively from \(M'\) to find a match. A matcher \(m\) is selected based on a subset of properties from \(P = p_1, p_2,…, p_k\), where \(k\) is the number of properties. Some property depends on term lengths and some on the distance function as used in constraint matcher. Following algorithm describes the whole column name assignment process.

Procedure AnnotateColumn
Input: ExpectedColumnNames N, and ColumnVector C.
Output: Labeled ColumnVector C
begin
1 \(R = \text{ExtractStructure}(C)\)
2 For each element \(i ∈ R\)
3 Search Ontology O
4 Extract possible Label l
5 For each element \(C_j\) in C
6 Extract Best Ranked label \(l_r = \text{OntoMatch}(l, C_j)\)
7 if \(l_r > l_i\)
8 Assign l to \(i_h\) Column
9 else
10 Prompt the user for a new Column name
11 Update O
12 Return C
end

If we take a look at an example of a web site containing information about name and address and salary information of the employees of an organization, the column name annotation scenario will be transparent. The table contains...
the following columns, Employee Name, Street, City, Zip, and Salary as shown in the table below.

<table>
<thead>
<tr>
<th>Name</th>
<th>Street</th>
<th>City</th>
<th>Zip</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Robert</td>
<td>4500 Cass Ave</td>
<td>Detroit</td>
<td>48207</td>
<td>$20000k</td>
</tr>
<tr>
<td>Moore</td>
<td>919 Rupert St</td>
<td>Chicago</td>
<td>32071</td>
<td>$25000k</td>
</tr>
<tr>
<td>Jen Doe</td>
<td>2323 Oaktrail Dr.</td>
<td>Bloomfield</td>
<td>48292</td>
<td>$50000k</td>
</tr>
</tbody>
</table>

From the above information, the structural pattern extractor will extract the following patterns:

Name: [s]+
Street: [n]+[s]+
City: [s]+
Zip: [n]+[s]+
Salary: $[n]+k

Thus the concept Address will consist of the sub-concepts Street, City and Zip and the sub-concept Zip will point to the knowledgebase KB:STATE for the state names and their acronyms. After processing several pages, the knowledgebase will be enriched and the address concept can be represented by the following rule;

Address ::= Street Sym City Sym Zip | Street Sym City Sym State Sym Zip
City ::= [s]+
Street ::= [n]+[s]+
Zip ::= [n]+ | [n]+[s]+
State ::= [s]+
Sym ::= [,\s.]*

The address field, however, may contain several other regular expressions as well depending on the variants of address representations in different countries. Suppose the user is interested to extract all the address information from the following un-annotated address:

John 31AB433 455 woodward st, Detroit, MI 48202
Jane 97C2331 311 Schafer Road, Dearborn, MI 48322

Here we assume that the the second column of the table represents some sort of ID internal to an organization. The second and the third field will match with the address concept and the identifier concept (assuming that Identifier concept is present in the ontology) and the first field will match with several sub-concepts e.g. city, state, etc. But since the user expectation is to extract location, the OntoMatch component will rank address as the best match for location. This is because

\[ \text{Rank}_{\text{address}} = \text{OntoMatch(address,location)}, \]
\[ \text{Rank}_{\text{identifier}} = \text{OntoMatch(address,location)}, \]
\[ \text{Rank}_{\text{address}} > \text{Rank}_{\text{identifier}}. \]

In order to label columns with label location/address we have two candidate column vectors, hence to resolve dispute we consult the knowledgebases pertaining to identifier and address concepts and return the third column since it contains location related keywords e.g. state name and city name etc. Thus the whole column name assignment process works as the following:

- First the actual table of interest is extracted using FastWrap.
- Each column of the table is scanned for structural pattern extraction, clues from data and their structures are subsequently used to form a set of candidate annotations from the ontology. This candidate set is refined using the query variables that is looking for specific attributes. OntoMatch is employed to prune the set of candidate column names.
- The concepts are analyzed to see if they match a higher order concept thereby eliminating more candidates. In this case visually adjacent columns are grouped together to see whether they match a higher order concept.
- At each phase we refine the ontology to accommodate variants of a concept. For example address field may or may not contain an apartment subfield thus both variants are included in the ontology.

4 Summary and Future Research

In this paper, we presented a novel wrapper generation system with automatic column name identification capability. The performance of the column name identifier improves monotonically as the number of web sites analyzed is increased. Thus as the ontology and the knowledge base is enriched, the assigned column names will continue to be more accurate and efficient. We regard our proposed technique as a first step toward autonomous annotation of missing column names that warrants more research.

References
