Table Interpretation by Sibling Page Comparison

Cui Tao and David W. Embley
Department of Computer Science
Brigham Young University Provo, UT, 84602, USA
cctao,embley@cs.byu.edu

Abstract

The longstanding problem of automatic table interpretation still illudes us [23]. Its solution would not only be an aid to table processing applications such as large volume table conversion and information extraction, but would also be an aid in solving related problems such as ontology learning [19] and semi-structured data management [11]. In this paper, we offer a solution for the common special case in which so-called sibling pages are available. Sibling pages, which are the pages commonly generated by underlying web databases, are compared to identify and connect nonvarying components (category labels) and varying components (data values). We tested our solution using more than 2,000 tables in source pages from three different domains—car advertisements, molecular biology, and geopolitical information. Experimental results show that the system can successfully identify sibling tables, generate structure patterns, interpret different tables using the generated patterns, and automatically adjust the structure patterns as needed.

1 Introduction

Tables present information in a simplified and compact way in rows and columns. Data in one row/column usually belongs to the same category or provides values for the same concept. The labels of a row/column describe this category or concept.

Although a table with a simple row and column structure is common, tables can be much more complex. Figure 1 shows an example. Tables may be nested or conjoined as are the tables in Figure 1. Labels may span across several cells to give a general description as does Identification and Location in Figure 1. Although labels commonly appear on the top or left, table designers occasionally place labels on the right side of a table. In long tables, labels sometimes appear at the end of a table or in the middle of a table, every few rows, in order to help a reader find the correspondence between labels and data. Sometimes tables are rearranged to fit the space available. Label-value pairs may appear in multiple columns across the pages or in multiple rows placed below one another down the page. These complexities make automatic table interpretation a challenging task.

To interpret a table is to properly associate table category labels with table data values. Using Figure 1 as an example, we see that Identification, Location, and Function are labels for the large rectangular table. Inside the right cell of the first row is another table with headers IDs, NCBI KOGs, Species, etc. Nested inside of this cell are two tables with labels CGC name, Sequences name, Other name(s), WB Gene ID, Version, and Gene Model, Status, Nucleotides(coding/transcript), Protein, and Amino Acids. Most of the rest of the text in the large rectangular table comprises the data values. If we look more closely, however, we may conclude that some category labels are interleaved in the text. For example, via person appears to be a label under CGC name, as does Entrez Genes and Ace View beside NCBI.

Once category labels and data values are found, we want to properly associate them. For example, the associated label for the value F18H3.5 should be the sequences of labels Identification, IDs, and Sequence name. Given the source table in Figure 1, we match category labels with values as Figure 2 shows. We associate one or more sequences of labels with each data value in a table. Borrowing notation from Wang [20], the left hand side of the arrow is a sequence of one or more table labels, and the right hand side of the arrow is a data value. For the first three label-value pairs in Figure 2, there is only one label sequence. The fourth, however, has two: Identification.Gene model(s).Gene model and 1. Each label sequence represents a dimension. In general, a table may have one, two, three, or more dimensions. If a table has multiple records (usually multiple rows) and if the records do not have labels, we add record numbers. The table under Identification.Gene model(s), for example, has two records (two rows), but no row labels. We therefore label the first record 1 and the second record 2. Thus, the label-value association becomes
Gene Summary for cdk-4

Specify a gene using a gene name (e.g., cdk-4), a predicted gene ID (e.g., P12945), or a protein ID (e.g., CE01636). Then click the search button:

<table>
<thead>
<tr>
<th>Gene Summary</th>
<th>Locus Summary</th>
<th>Sequence Summary</th>
<th>Protein Summary</th>
<th>EST Summary</th>
<th>Genome Browser</th>
<th>Genetic Maps</th>
<th>Submit</th>
<th>Searches</th>
<th>Site Map</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Identification.IDs.CGC name</th>
<th>Identification.IDs.Sequence name</th>
<th>Identification.Gene model(s).Amino Acids, 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>cdk-4 (Cyclin-Dependent Kinase family) (via person: Michael Krause)</td>
<td>F18H3.5</td>
<td>406 aa</td>
</tr>
</tbody>
</table>

(NCBI KOGs: Protein kinase PT/TA/ES and related kinases [KOG0094])
Species: Caenorhabditis elegans
Other sequence(s): AP003778 (Caenorhabditis elegans cyclin-dependent kinase CDK-4 (cdk-4) mRNA, complete cds.)
NCBI: (BroadGen: 15716266) [AceView: X0136]

Gene model(s):
| Putative orthology(s): | Caenorhabditis briggsae: CE007433 (protein fragment) [Shin JD et al., best reciprocal blast match-seq-ELS] |

<table>
<thead>
<tr>
<th>Identification.Gene model(s).Gene model(s).Gene Model</th>
<th>Status</th>
<th>Nucleotides (coding/transcript)</th>
<th>Protein</th>
<th>Amino Acids</th>
</tr>
</thead>
<tbody>
<tr>
<td>F18H3.5 s1, 2</td>
<td>confirmed by cDNA</td>
<td>12098351 bp</td>
<td>WP:CE01636</td>
<td>342 aa</td>
</tr>
<tr>
<td>F18H3.5 s1, 2, 3</td>
<td>partially confirmed by cDNA</td>
<td>12217846 bp</td>
<td>WP:CE02918</td>
<td>486 aa</td>
</tr>
</tbody>
</table>

Location:
<table>
<thead>
<tr>
<th>Generic Position:</th>
<th>X:12.69 +/- 0.000 cm [mapping data]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generic Position:</td>
<td>X:12918822.1, 13917773 bp</td>
</tr>
</tbody>
</table>

Function:
<table>
<thead>
<tr>
<th>Mutant Phenotype:</th>
<th>(Figure MTF) cdk-4 is a cycline-dependent kinase related to cdk-4 and cdk-6 from other organisms. Homozygous cdk-4(-/-) animals usually arrest in L3 due to so, or limited, proliferation of the post-embyronic blast cells. About 7% of animals make it to a late stage of development. Definitions of abbreviations used in the text.</th>
</tr>
</thead>
<tbody>
<tr>
<td>RNAi Phenotype(s):</td>
<td>let-70 function (For details see: Park M 06 Oct 1999)</td>
</tr>
</tbody>
</table>

Figure 1. A sample page from WormBase [1].

Figure 2. Interpretation for the tables in Figure 1 (partial).

(Identification.IDs.CGC name) → cdk-4 (Cyclin-Dependent Kinase family)
(via person: Michael Krause);
(Identification.IDs.Sequence name) → F18H3.5;
(Identification.Gene model(s).Amino Acids, 2) → 406 aa;

To interpret a table automatically, especially a table as complex as the one in Figure 1, is not an easy task. If we have another page, such as the one in Figure 3, that has essentially the same structure, the system might be able to obtain enough information about the structure to make automatic interpretation possible. We call pages that are from the same website and have similar structures sibling pages. The two pages in Figures 1 and 3 are sibling pages. They have the same basic structure, with the same top banners that appear in all the pages from this website, with the same table title (Gene Summary for some particular gene), and a table that contains information about the gene. Corresponding tables in sibling pages are called sibling tables. If we compare the two large tables in the main part of the sibling pages, we can see that the first columns of each table are exactly the same. If we look at the cells under the Identification label in the two tables, both contain another table with two columns. In both cases, the first column contains identical labels IDs, NCBI KOGs, ..., Putative orthology. Further, the tables under Identification.IDs also have identical header rows. The data rows, however, vary considerably. General speaking, we can look for commonalities to find labels and look for variations to find data values.

Given that we can find most of the label and data cells in this way, our next task is to infer the general structure pattern of the website and of the individual tables embedded within pages of the website. With respect to identified labels, we look below and/or to the right for the right value associations; we may also need to look above and/or to the left. In Figure 1, the values for Identification.Gene Model(s).Gene Model are below, and the values for Identification.Species are to the right.

Although we look for commonalities to find labels and look for variations to find data values, we must be careful
Gene Summary for dyb-1

Specify a gene using a gene name (e.g. 20), a predicted gene ID (CB02711), or a protein ID (CB02711) dyb-1

<table>
<thead>
<tr>
<th>Identification</th>
<th>Sequence name</th>
<th>Other name(s)</th>
<th>WB Gene ID</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>dyb-1 - C[(\alpha)tubulin homologue] (via person: Laurens Teunissen)</td>
<td>E4756.1</td>
<td>IBM-45452 (altered automatically)</td>
<td>13963 (altered automatically)</td>
<td>WB:Gene:0001115</td>
</tr>
</tbody>
</table>

The dyb-1 gene encodes a homolog of mammalian alpha-d/etubulin (CT19M, OMIM:603291): mutation of which can lead to left ventricular noncompaction with congenital heart defects. [details]

NCBI KO IDs: Beta-tubulin [KO:3480]

C. elegans line: [E4756.1] [AcetView:13963]

<table>
<thead>
<tr>
<th>Gene model(s):</th>
<th>Gene Model</th>
<th>Status</th>
<th>Nucleotides (coding/transcript)</th>
<th>Protein</th>
<th>Swissprot</th>
<th>Amino Acids</th>
</tr>
</thead>
<tbody>
<tr>
<td>dyb-1 - 1.2</td>
<td>confirmed by (3DAAA)</td>
<td>1777101 bp</td>
<td>WPC-02507</td>
<td>ETH-CAREL</td>
<td>220 aa</td>
<td></td>
</tr>
</tbody>
</table>

C. elegans line: [E4756.1] [AcetView:13963] [3DAAA]

Mutant Phenotypes: Definitions of abbreviations used in the text.

RNAi Phenotype(s): WT [For details see: Abzyan JA, 11 Nov 2000]

Figure 3. A second sample page from WormBase [1].

about being too strict. Sometimes there are additional or missing label-value pairs. The two nested tables beginning with Gene Model in Figures 1 and 3 do not share exactly the same structure. The table in Figure 1 has five columns and three rows, while the table in Figure 3 has six columns and two rows. Although they are not exactly the same, we can still identify the structure pattern by comparing them. The top rows in the two tables are very similar. Observe that the table in Figure 3 only has an additional Swissprot column inserted between the Protein and Amino Acids columns. Although the labels for the two tables are not identical, we can still tell that they are table headers.

In addition to discovering the structure pattern for a web site, we can also dynamically adjust the pattern if the system encounters a table that varies from the pattern. If there is an additional or missing label, the system can change the pattern by either adding the new label and marking it optional or marking the missing label optional. For example, if we had not seen the extra Swissprot column in our initial pair of sibling pages, the system can add Swissprot as a new label and mark it as optional. The basic label-value association pattern is still the same.

Other researchers have also tried to take advantage of sibling pages. RoadRunner [7] compares two HTML pages from one web site and analyzes the similarities and dissimilarities between them in order to generate extraction wrappers. It discovers data fields by string mismatches and discovers iterators and optionals by tag mismatches. RoadRunner, however, only focuses on finding data fields. It does not discover labels or try to associate data and labels. Our system focuses on table interpretation. It looks for a table pattern in addition to data fields. Furthermore, our system also dynamically adjusts the structure pattern as new, as yet unseen, structures are encountered.

Recent surveys [10, 23] describe the vast amount of research that has been done in table processing and illustrate the challenges of the table interpretation problem. We focus in this paper, however, only HTML tables. A number of HTML table extraction systems use machine learning to recognize tables in web pages (e.g. [6, 21]). Drawbacks of machine learning approaches, however, are that they need training data, and they need to be retrained for tables from different web sites. Other table interpretation systems work based on some simple assumptions and heuristics (e.g. [5, 9, 15]). These simple assumptions (labels are either the first row or the first column) are easily broken in complex tables. More sophisticated table interpretation techniques have appeared in recent papers [13, 14, 16]. None of this research makes use of sibling tables, but is complementary to our work and could potentially be used in conjunction with our work in future efforts to improve results for certain cases.
We present the details of our contribution to table interpretation by sibling page comparison in the remainder of the paper as follows. Section 2 provides the details about how the system analyzes a source page to recognize all HTML tables and how it decomposes nested tables, if any. Section 3 introduces the matching algorithms we use. Section 4 describes how we interpreted various matching results. Section 5 explains how the system infers the general structure patterns of a web site, and therefore how to interpret the tables from this site. It also explains how to automatically adjust the generated patterns when variations are encountered. In Section 6, we report the results of experiments we conducted involving sites for car advertisements, molecular biology, and geopolitical information, which we found on the web. In Section 7, we draw conclusions and mention some possibilities for future work.

2 Table Recognition

The tags `<table>` and `</table>` delimit HTML tables in a web document. In each HTML table, there may be tags that specify the structure of the table. The tag `<th>` is designed to declare a header, `<tr>` is designed to declare a row, and `<td>` is designed to declare a data entry. Unfortunately, we cannot count on users to consistently apply these tags as they were originally intended. Most table designers simply use the `<td>` tag for every table entry without regard to whether it is a header or a data value. In addition, a web page designer might (1) use table tags for layout (i.e. to line up columns and rows of symbols, or values, or statements with no thought of table headers, values, and their associations), or (2) not use HTML tags to represent a table (i.e. use verbatim layout of symbols, values, and statements to form a table). For the first case, our system needs to determine that the object delimited by HTML table tags is not a table. For the second case, the solution requires techniques beyond those discussed in this paper. We consider this to be interesting future research.

After obtaining a source document, the system first parses the source code and locates all HTML components enclosed by `<table>` and `</table>` tags (tagged tables). In Figure 1, for example, there are many tagged tables. The upper banner with Home, Genome, etc. is a tagged table. The lower banner with Gene Summary, Locus Summary, etc. is another tagged table. Yet another tagged table is the page index: [identification] [location] [function] ... . The main section of this page enclosed in a rectangle that starts with Identification is another tagged table. There are tagged tables nested inside of this table. A `</table>` tag appears before IDs; it’s corresponding end tag appears after (Stein LD et al. best reciprocal blastp match-seg-off). The two nested rectangular components immediately to the right of IDs and Gene model(s) respectively also have enclosing `<table>`-tags. In addition, the two rectangular cells to the right of Location and Function are tagged tables too.

When tagged tables are nested inside of one another, the system finds them and unnests them. In Figure 1, there are several levels of nesting in the large rectangular table. The first level is a table with two columns. The first column contains Identification, Location, and Function; and the second column contains some complex structures. Figure 1 shows only the first three rows of this table—one row for Identification, one for Location, and one for Function. (For the purpose of being explicit in this paper, we assume that these three rows are the only rows in this table.) The second column of the large rectangular table in Figure 1 contains three second level nested tables, the first starting with IDs, the second with Genetic Position, and the third with Mutant Phenotype. In the right most cell of the first row is another table. In this table, there are two third level nested tables.

We treat each tagged table as an individual table and assign an identifying number to it. If the table is nested, we replace the table in the upper level with its identifying number. By so doing, we are able to remove nested tables from upper level tables. As a result, the system decomposes the page in Figure 1 into the set of tables in Figure 4.
Figure 5. DOM trees for Table 7 in Figure 4 and its sibling table in Figure 3.

3 Table Matching

To compare and match tables, we first transform each HTML table into a DOM tree [8]. Tree1 in Figure 5 shows the DOM tree for Table 7 in Figure 4, and Tree2 in Figure 5 shows the DOM tree for its corresponding table in Figure 3.

There are a number of algorithms for comparing different kinds of trees [4]. In our specific case, we are only interested in labeled, ordered, rooted trees. A rooted tree is a tree whose root vertex is fixed. The fixed root vertex of a DOM tree of an HTML table is always a table tag. Ordered rooted trees are root trees in which the relative order of the children is fixed for each vertex. The HTML tag path as well as the relative order of the children in each branch help determine the layout of a table and the position of each cell. Labeled ordered rooted trees have a label attached to each of their vertices. In an HTML DOM tree, each node has a label that is either a tag (for internal tree nodes) or a cell’s content (for leaf nodes).

Tai [17] gives a well acknowledged formal definition of the concept of a tree mapping for labeled ordered rooted trees.

Let $T$ be a labeled ordered rooted tree and let $T[i]$ be the $i^{th}$ node in level order of tree $T$. A mapping from tree $T$ to tree $T'$ is defined as a triple $(M, T, T')$, where $M$ is a set of ordered pairs $(i, j)$, where $i$ is from $T$ and $j$ is from $T'$, satisfying the following conditions:

1. $i_1 = i_2$ iff $j_1 = j_2$, where $i_1$ and $i_2$ are two nodes from $T$ and $j_1$ and $j_2$ are two nodes from $T'$;
2. $T[i_1]$ comes before $T[i_2]$ iff $T'[j_1]$ comes before $T'[j_2]$ in level order;
3. $T[i_1]$ is an ancestor of $T[i_2]$ iff $T'[j_1]$ is an ancestor of $T'[j_2]$.

According to this definition, each node appears at most once in a mapping and the order between sibling nodes and the hierarchical relation between nodes are preserved. The best match between two trees is a mapping with the maximum number of ordered pairs.

We use a simple tree matching algorithm introduced in [22] which was first proposed to compare two computer programs in software engineering. It calculates the similarity of two trees by finding the best match through dynamic programming with complexity $O(n_1n_2)$, where $n_1$ is the size (number of nodes) of $T$ and $n_2$ is the size of $T'$. This algorithm counts the matches of all possible combination pairs of nodes from the same level, one from each tree, and finds the pairs with maximum matches. The simple tree match algorithm returns the number of these maximum matched pairs. Figure 6 shows the matched nodes for $Tree_1$ with respect to $Tree_2$ in Figure 5. The highlighted nodes indicate a match.
4 Sibling Table Identification

In our research, we use the results of the simple tree matching algorithm for three tasks: (1) we filter out those HTML tables that are only for layout; (2) we identify the corresponding tables (sibling tables) from sibling pages; and (3) we match nodes in a sibling table pair.

For each pair of trees, we use the simple tree matching algorithm to find the maximum number of matched nodes among the two trees. We call this number the match score. For each table in one source page, we obtain match scores and thus a ranking for all tables in a sibling page. Sibling tables should have a one to one correspondence. Based on the match score, we use the Gale-Shapley stable marriage algorithm [12] to pair potential sibling tables one to one.

For a pair of potential sibling tables, we calculate the sibling table match percentage, 100 times the match score divided by the number of nodes of the smaller tree. The match percentage between the two trees in Figure 5, for example, is 19 (match score) divided by 27 (tree size of $Tree_2$), which, expressed as a percentage, is 70.4%.

We classify potential sibling tables into three categories: (1) exact match or near exact match; (2) false match; and (3) sibling-table match. We use two threshold boundaries to classify potential sibling tables: a higher threshold between exact or near exact match and sibling-table match, and a lower threshold between sibling-table match and false match. Usually a large gap exists between the range of exact or near exact match percentages and the range of sibling-table match percentages, as well as between the range of sibling-table match percentages and the range of false match percentages. Using active learning with bootstrap selective sampling [18], we first set initial thresholds by empirical observation (90% for the higher threshold and 20% for the lower threshold); then the system dynamically adjusts the two thresholds as needed during the classification process as more sibling pages are considered.

In our example, Tables 1, 2, and 3 have match percentages of 100% with their sibling tables. The match percentages for Tables 4, 5, 6 and 7, and their corresponding sibling tables, are 66.7%, 58.8%, 69.2%, and 70.4% respectively.

5 Structure Patterns

The first component of a structure pattern for a table specifies the table’s location in a web page. To specify the location, we use XPath [3], which describes the path of the table from the root HTML tag of the document. For example, the location for Table 7 in Figure 4 is:

```
```

An XPath simply lists the nodes (HTML tag names) of a path in DOM tree for the HTML document where [n] designates the nth sibling node in the ordered subtree.

The second component of a structure pattern specifies the label-value pairs for a table and thus provides the interpretation. We now give the details about how we identify the proper label-value pattern template (Section 5.1) and use it to generate the specific label-value-pair pattern for the table (Section 5.2). We then explain how the system uses the generated pattern to extract label-value pairs from the table and how the system produces Wang notation and thus provides an interpretation for the table (Section 5.3). Combinations of basic patterns are also possible; we thus also explain how to generate and use combination patterns (Section 5.4). Finally, we explain how the system dynamically adjusts a pattern to accommodate table variations it may encounter as it extracts label-value pairs from sibling tables in the web site.

5.1 Pattern Templates

We use regular expression to describe table structure pattern templates. If we traverse a DOM tree, which is ordered and labeled, in a preorder traversal, we can layout the tree labels textually and linearly. We can then use regular-expression like notation to represent the table structure patterns (see Figure 7). In both templates and generated patterns we use standard notation: ? (optional), + (one or more repetitions), and | (alternative). In templates, we augment the notation as follows. A variable (e.g. $n$) or an expression (e.g. $n-1$) can replace a symbol to designate a specific number of repetitions, which is unknown but fixed for the expression as it is applied. A pair of braces ( { } ) indicates a leaf node. A capital letter $L$ is a position holder for a label and a capital letter $V$ is a position holder for value. The part in a box is an atomic pattern which we use for combinational structural patterns in Section 5.4.

Figure 7 shows three basic pre-defined pattern templates. Pattern 1 is for tables with $n$ labels in the first row and with $n$ values in each of the rest of the rows. The association between labels and values is column-wise; the label at the top of the column is the label for all the values in each column.

Pattern 2 is for tables with labels in the left-most column and values in the rest of the columns. Each row has a label followed by $n$ values. The label-value association is row-wise; each label labels all values in the row.

Pattern 3 is for two-dimensional tables with labels on both the top and the left. Each value in this kind of table associates with both the row header label and the column header label.
Figure 7. Some basic pre-defined pattern templates.

5.2 Pattern Generation

To check whether a table matches any pre-defined pattern template, the system tests each template until it finds a match. When we search for a matching template, we only consider leaf nodes and seek matches for labels and mismatches for values. Variations, however, exist and we must allow for them. Specifically, we calculate a template match percentage between a pre-defined template and a matched result, 100 times the number of leaf nodes that agree with a pattern template divided by total number of leaf nodes in the tree. We calculate the template match percentage between a table and each pre-defined structure template. A match must satisfy two conditions: (1) it must be the highest match percentage, and (2) the match percentage must be greater than a threshold. Similar to the way we determine thresholds for sibling table matches, we determine this template match percentage threshold using active learning with bootstrap selective sampling, with an initial threshold of 80%.

Consider the mapped result in Figure 6 as an example. The highlighted nodes are matched nodes. Comparing the template match percentage for this mapped result for the three pattern templates in Figure 7, we obtain 93.3%, 53.3%, and 80% respectively. Pattern 1 has the highest match percentage, and it is greater than the threshold. Therefore we choose Pattern 1.

We now impose the chosen pattern, ignoring matches and mismatches. Note that for the tree in Figure 6, the first branch matches the part in Pattern 1 in the first box and the second and the third branch, each match the part in the second box, where n is five. For Pattern 1, when n=1, we have a one-dimensional table; and when n>1, we have a two-dimensional table for which we must generate record numbers.

After the system matches a table with a pre-defined pattern template, it generates a specific structure pattern for the table by substituting the actual labels for each L and by substituting a placeholder V for each value. The subscript L for a value V designates the label-value pair for each record in a table. Figure 8 shows the specific structure pattern for Table 7 in Figure 4.

5.3 Pattern Usage

With a structure pattern for a specific table, we can interpret the table and its sibling tables. The XPath gives the location of the table, and the generated pattern gives the label-value pairs. The pattern must match exactly in the sense that each label string encountered must be identical to the pattern’s corresponding label string. Any failure is reported to the system. (In Section 5.5, we explain how the system reacts to a failure notification.)

When the pattern matches exactly, the system can generate Wang notation and thus can provide an interpretation for the table. For our example, the chosen pattern is Pattern 1 with + (which allows for multiple rows of values in the table). Thus, the system needs to add another dimension and add row numbers. Since the table is inside of other tables, the system recursively searches for the tables in the upper levels of nesting and collects all needed labels. Figure 9 shows Table 7 in Figure 4 in Wang notation.
Figure 10. An example for pattern combination from MutDB [2].

Example 1:

\[
\begin{align*}
\text{table} & : (\text{table})^+ \\
\text{tr} & : (\text{td}(\text{th}) \uparrow \{L\} (\text{td}(\text{th}) \uparrow \{V\})^*)^+ \\
\text{tr} & : (\text{td}(\text{th}) \uparrow \{L\} (\text{td}(\text{th}) \uparrow \{V\})^m (\text{tr} \uparrow \{L\} (\text{td}(\text{th}) \uparrow \{V\})^m)^+ \\
\end{align*}
\]

Example 2:

\[
\begin{align*}
\text{table} & : (\text{table})^+ \\
\text{tr} & : (\text{td}(\text{th}) \uparrow \{L\} (\text{td}(\text{th}) \uparrow \{V\})^n \\
& \quad \text{td}(\text{th}) \uparrow \{L\} (\text{td}(\text{th}) \uparrow \{V\})^m)^+ \\
\end{align*}
\]

Figure 11. Two examples of pattern combinations.

5.4 Pattern Combinations

It is possible that the system cannot match any pre-defined template. In this case, it looks for pattern combinations. Using Figure 10 as an example, assume that the system matches the all cells in the first and third column, but none in the second and forth column. Comparing the template match percentage for this mapped result for the three pattern templates in Figure 7, we obtain 50%, 75%, and 68.8% respectively. None of them is greater than the threshold, 80%. The first two columns, however, match Pattern 2 perfectly, as do the last two column.

In many cases, tables can be more complicated. Most complex tables do not match to only one pre-defined pattern template, but do match to a combination of several of them. Patterns can be combined row-wise or column-wise. In a row-wise combination, one pattern template can appear after another, but only the first pattern template has the header: \( < \text{table} > (\langle \text{tbody} \rangle)^+ \). Therefore, a row-wise combined structure pattern has a few rows matching one template and other rows matching another template. In a column-wise combination, we combine different atomic patterns. If a pattern template has two atomic patterns, both patterns must appear in the combined pattern, in the same order, but they can be interleaved with other atomic patterns. If one atomic pattern appears after another atomic pattern from a different pattern template, the \( < \text{tr} > \) tag at the beginning is removed. Figure 11 shows two examples of pattern combinations. Example 1 combines Pattern 2 and Pattern 1 row-wise. Example 2 combines Pattern 2 with itself column-wise. This second pattern matches the table in Figure 10, where \( n = m = 1 \), and the plus (+) is 4.

The initial search for combinations is similar to the search for single patterns. The system checks patterns until it finds mismatches, it then checks to see whether the mismatched part matches with some other pattern. The system first searches row-wise for rows of labels and then uses these rows as delimiters to divide the table into several groups. If it cannot find any row of labels, it repeats the same process column-wise. The system then tries to match each sub group with a pre-defined template. This process repeats recursively until all sub-groups match with a template.

For the example in Figure 10, the system is unable to find any rows of labels, but finds two columns of labels, the first and third column. It then divides the table into two groups using these two columns and tries to match each group with a pre-defined template. It matches each group with Pattern 2. Therefore, this table matches column-wise with two combinations of Pattern 2.

5.5 Dynamic Pattern Adjustment

Given a structure pattern for a table, we know where the table is in the source document (its XPath), the location of the labels and values, and the association between labels and values. When the system encounters a new sibling page, it tries to locate each sibling table following the XPath, and then to interpret it by matching it with the sibling table structure pattern. If the new encountered table matches the structure pattern regular expression perfectly, we successfully interpret this table. Otherwise, we might need to do some pattern adjustment. There are two ways to adjust a structure pattern: (1) adjust the XPath to locate a table, and (2) adjust the generated structure pattern regular expression.

Although sibling pages usually have the same base structure, some variations might exist. Some sibling pages might have additional or missing tables. Thus, sometimes, following the XPath, we cannot locate the sibling table for which we are looking. In this case, the system searches for tables at the same level of nesting, looking for one that matches the pattern. If the system finds one, it obtains the XPath and adds it as an alternative. Thus, for future sibling pages, the system can (in fact, always does) check all alternative XPaths before searching for another alternative XPath. If the system finds no matching table, it simply continues its processing with the next table.

We adjust a table pattern when we encounter a variation of an existing table. There might be additional or missing labels in the encountered variation. In this case, we need to adjust the structure pattern regular expression, to add the new optional label or to mark the missing label as optional. Consider the table that starts with Gene Model in Figure 3 (the sibling table of Table 7 in Figure 4) as an example. The table matches the pattern in Figure 8 until we encounter the
Figure 12. Structure Pattern for the table in Figure 3.

label Swissprot. If we skip Swissprot, the next label Amino Acids matches the structure pattern. In this case, we treat Swissprot as an additional label, and we add it as an optional label as Figure 12 shows.

6 Experimental Results

We tested the system using source pages from commercial data, scientific data, and world fact data. We picked one domain from each field: car advertisements, molecular biology, and geopolitical information. We tested more than 2,000 tables from 275 sibling pages in 35 web sites from these three domains. For each web site, we randomly chose two sibling pages for the initial pattern generation. For the initial two sibling pages, we tested whether the system was able to locate all HTML tables, whether it was able to pair all the sibling tables correctly, and whether it was able to recognize the correct pattern template or pattern combination. For the rest of sibling pages from the same web site, we tested whether the system was able to interpret tables using the recognized structure patterns, whether any interpretation needed dynamic adjustment, and whether the system recognized new structure patterns correctly.

We collected 75 sibling pages from 15 different web sites in the car advertisements domain for a total of 780 HTML tables. The system correctly discarded all uses of tables for layout and successfully paired all sibling tables. There were no nested tables in this domain. Two web sites contained tables with structure combinations. Most of the web sites contained only one table pattern, except for one site that had three different patterns. Of the 17 pairs of sibling tables, the system recognized 16 correctly. The one pattern the system failed to recognize correctly contained too many values, such as unknown, general car, auto, dealer, and empty spaces, that were the same. The system considered them as labels, and thus could not match the table with any predefined pattern template or detect any pattern combination. The system successfully interpreted all tables from the generated patterns. No adjustment were needed, neither for any path nor for any label.

For the geopolitical information domain, we tested 100 sibling pages from 10 different web sites with 884 HTML tables. The system paired 100% of the sibling tables correctly. For initial pattern generation, the system was able to recognize all 22 structure patterns successfully. As the system processed additional sibling pages, it found one additional sibling table and correctly interpreted it. There were no path adjustments, but there were 22 label adjustments—all of them correct. For two sets of sibling tables, the system recognized the correct patterns, but failed to recognize some implicit information that affects the meaning of the tables. Therefore it interpreted the tables only partially correctly. Figure 13 shows these two cases. There are actually two HTML tables in Figure 13a. The header Geography Mongolia is in one HTML table, and the rest of information is in another HTML table. Because we separated tables using HTML tags, the system was not able to determine the relationship between these two HTML tables. The system correctly interpreted Figure 13b as Pattern 3. It, however, did not catch the relationship between Murders and per 100,000 and between Rapes and per 100,000.

We collected 100 sibling pages from 10 different web sites in the molecular biology domain for a total of 862 HTML tables. Among these tables, the system falsely classified three pairs of layout tables as data tables. The system, however, successfully eliminated these false sibling pairs during pattern generation because it was unable to find a matching pattern. No false patterns were generated. The system was able to recognized 28 of 29 structure patterns. The system missed one pattern because the table contained
too many empty cells. If we had considered empty cells as mismatches, the system would have correctly recognize this pattern. As the system processed additional sibling pages, it found 5 additional sibling tables and correctly interpreted all but one of them. The failure was caused by labels that varied across sibling tables causing them in some cases to look like values. There were 5 path adjustments and 12 label adjustments—all of them correct. One table was interpreted as a header.

7 Conclusion and Future Work

By comparing tables in sibling pages, we are able to find the location of table labels and data entries, and pair them to infer the general pattern for all pages from the same site. Our experiments using source pages from three different domains—car advertisements, molecular biology, and geopolitical information—indicate that the system can succeed in properly interpreting tables in sibling pages.

In the system we introduce in this paper, we assumed that information in one table cell is either a table label or a table value. There could be structured information within a cell, however, such as the label via person; and the value Michael Krause in Figure 1. As a future work on table interpretation, we would like to analyze cell content to find structure information within cells. Furthermore, we would like to extend the system, so that it can work with non-HTML tables too.

References