Automatic Hidden-Web Table Interpretation by Sibling Page Comparison

Cui Tao[⋆] and David W. Embley[∗]

Brigham Young University, Provo, Utah 84602, U.S.A.
{ctao, embley}@cs.byu.edu

Abstract. The longstanding problem of automatic table interpretation still illudes us. Its solution would not only be an aid to table processing applications such as large volume table conversion, but would also be an aid in solving related problems such as information extraction and semi-structured data management. In this paper, we offer a conceptual modeling solution for the common special case in which so-called sibling pages are available. The sibling pages we consider are pages on the hidden web, commonly generated from underlying databases. We compare them 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 tables using the generated patterns, and automatically adjust the structure patterns, if necessary, as it processes a sequence of hidden-web pages. For these activities, the system was able to achieve an overall F-measure of 94.5%.

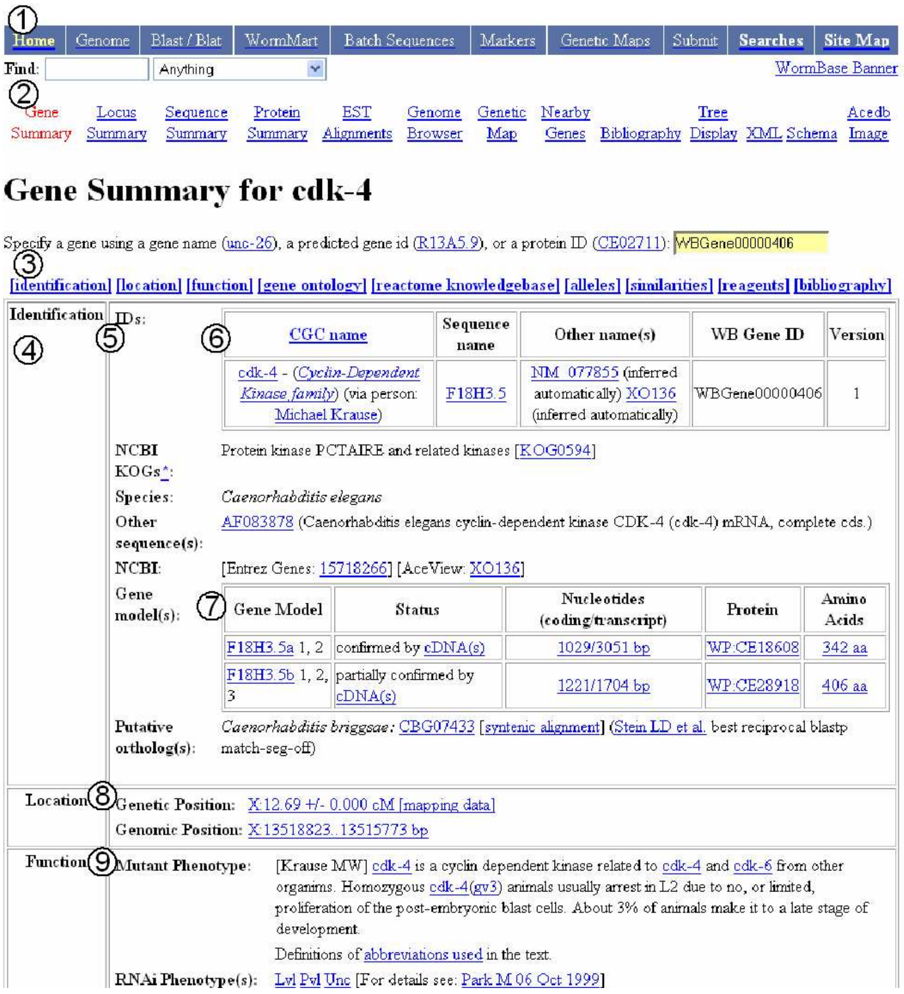
1 Introduction

The World Wide Web serves as a powerful resource for every community. Much of this online information, indeed, the vast majority, is stored in databases on the so-called hidden web.[1] Hidden-web information is usually only accessible to users through search forms and is typically presented to them in tables. Automatically understanding hidden-web pages is a challenging task. In this paper, we introduce a domain independent, web-site independent, unsupervised way to automatically interpret tables from hidden-web pages.

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.

[⋆] Supported in part by the National Science Foundation under Grant #0414644.

[1] There are more than 500 billion hidden-web pages. The surface web, which is indexed by common search engines only constitutes less than 1% of the World Wide Web. The hidden web is several orders of magnitude larger than the surface web [10].



①

Home	Genome	Blast / Blat	WormMart	Batch Sequences	Markers	Genetic Maps	Submit	Searches	Site Map

Find: Anything WormBase Banner

②

Gene Summary	Locus Summary	Sequence Summary	Protein Summary	EST Alignments	Genome Browser	Genetic Map	Nearby Genes	Bibliography	Tree Display	XML Schema	Acedb Image

Gene Summary for cdk-4

Specify a gene using a gene name (unc-26), a predicted gene id (R13A5.9), or a protein ID (CE02711): WBGene00000406

③

[identification] [location] [function] [gene ontology] [reactome knowledgebase] [alleles] [similarities] [reagents] [bibliography]

Identification ④

IDs: ⑤ ⑥

CGC name	Sequence name	Other name(s)	WB Gene ID	Version
cdk-4 - (*Cyclin-Dependent Kinase family*) (via person: Michael Krause)	F18H3.5	NM_077855 (inferred automatically) XO136 (inferred automatically)	WBGene00000406	1

NCBI KOGs*: Protein kinase PCTAIRE and related kinases [KOG0594]

Species: *Caenorhabditis elegans*

Other sequence(s): AF083878 (Caenorhabditis elegans cyclin-dependent kinase CDK-4 (cdk-4) mRNA, complete cds.)

NCBI: [Entrez Genes: 15718266] [AceView: XO136]

Gene model(s): ⑦

Gene Model	Status	Nucleotides (coding/transcript)	Protein	Amino Acids
F18H3.5a 1, 2	confirmed by cDNA(s)	1029/3051 bp	WP:CE18608	342 aa
F18H3.5b 1, 2, 3	partially confirmed by cDNA(s)	1221/1704 bp	WP:CE28918	406 aa

Putative ortholog(s): *Caenorhabditis briggsae:* CBG07433 [syntenic alignment] (Stein LD et al. best reciprocal blastp match-seg-off)

Location ⑧ **Genetic Position:** X:12.69 +/- 0.000 cM [mapping data]

Genomic Position: X:13518823..13515773 bp

Function ⑨ **Mutant Phenotype:** [Krause MW] cdk-4 is a cyclin dependent kinase related to cdk-4 and cdk-6 from other organims. Homozygous cdk-4(gv3) animals usually arrest in L2 due to no, or limited, proliferation of the post-embryonic blast cells. About 3% of animals make it to a late stage of development.

Definitions of abbreviations used in the text.

RNAi Phenotype(s): Lvl Pvl Unc [For details see: Park M 06 Oct 1999]

Fig. 1. A Sample Table from WormBase (http://www.wormbase.org).

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 do *Identification* and *Location* in Table 4 of 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 a page or in multiple rows placed below one another down a page. These complexities make automatic table interpretation challenging.

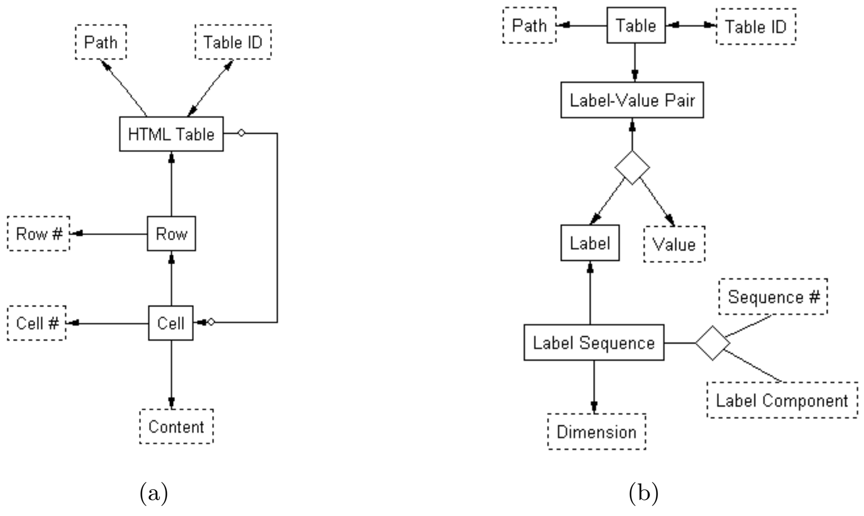
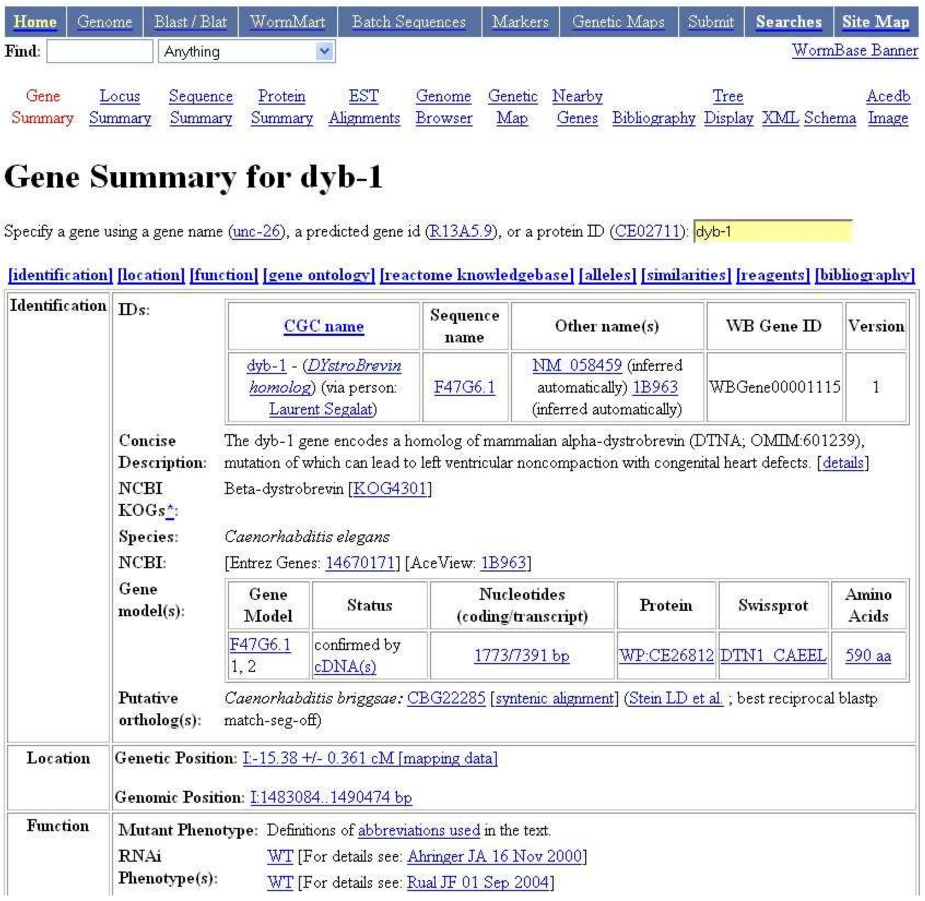


Fig. 2. Conceptual Model Instances for (a) An Input HTML Table and (b) An Output Interpretation.

In this paper, we introduce a conceptual-modeling-based table interpretation system. We use a conceptual-modeling language to model both the input tables and the output interpretations as suggested in [5]. When we do, the table-interpretation problem becomes a problem of transforming one populated conceptual-model instance to another.

Figure 2(a) shows the model instance for an HTML page containing one or more tables. Each HTML table has a unique *Table ID* (e.g. the table numbers in Figure 1), and a unique *Path* in terms of the page's DOM tree (e.g. /html/table[4]/tbody/tr[1]/td[2]/table[1]/tbody/tr[6]/td[2]). The tags <table> and </table> delimit HTML tables in a web document. An *HTML Table* has one or more *Row*s (delimited by <tr> tags); each *Row* has a *Row#*. A *Row* has one or more *Cell*s (delimited by <td> or <th> tags); each *Cell* has a *Cell#*. Each *Cell* contains *Content* and may contain other *HTML Table*s. The *Content* is the content of a cell, not between <table> and </table> tags. The *Content* may consist of HTML tags, images, and strings. Using Figure 1 as an example, Table 4 has three *Row*s, starting with *Identification*, *Location*, and *Function*. In the first *Row* are two *Cell*s. The *Content* of the first *Cell* is the string "*Identification*", and the content of the second *Cell* is an *HTML Table*, Table 5, which has seven *Row*s and fourteen *Cell*s, two of which contain tables, Table 6 and Table 7.

As Figure 2(b) shows, to interpret an HTML table is to properly associate table category labels with table data values, as the set of label-value pairs of the table. The *Path* for a table's interpretation is its path in an HTML page. We model the label-value pairs according to Wang notation [14]. The Wang Notation for a table is a set of *Label-Value Pair*s. Each *Label-Value Pair* contains one *Label* and one *Value*. Each *Label* has one or more *Label Sequence*s, one to describe



Home | Genome | Blast / Blat | WormMart | Batch Sequences | Markers | Genetic Maps | Submit | Searches | Site Map

Find: Anything   WormBase Banner

Gene Summary | Locus Summary | Sequence Summary | Protein Summary | EST Alignments | Genome Browser | Genetic Map | Nearby Genes | Bibliography | Tree Display | XML Schema | Acedb Image

Gene Summary for dyb-1

Specify a gene using a gene name (unc-26), a predicted gene id (R13A5.9), or a protein ID (CE02711): dyb-1

[identification] [location] [function] [gene ontology] [reactome knowledgebase] [alleles] [similarities] [reagents] [bibliography]

Identification

IDs:

CGC name	Sequence name	Other name(s)	WB Gene ID	Version
dyb-1 - (*DYstroBrevin homolog*) (via person: Laurent Segalat)	F47G6.1	NM_058459 (inferred automatically) 1B963 (inferred automatically)	WBGene00001115	1

Concise Description: The dyb-1 gene encodes a homolog of mammalian alpha-dystrobrevin (DTNA; OMIM:601239), mutation of which can lead to left ventricular noncompaction with congenital heart defects. [details]

NCBI KOGs*: Beta-dystrobrevin [KOG4301]

Species: *Caenorhabditis elegans*

NCBI: [Entrez Genes: 14670171] [AceView: 1B963]

Gene model(s):

Gene Model	Status	Nucleotides (coding/transcript)	Protein	Swissprot	Amino Acids
F47G6.1 1, 2	confirmed by cDNA(s)	1773/7391 bp	WP:CE26812	DTN1_CAEEL	590 aa

Putative ortholog(s): *Caenorhabditis briggsae:* CBG22285 [syntenic alignment] (Stein LD et al. ; best reciprocal blastp match-seg-off)

Location

Genetic Position: I:-15.38 +/- 0.361 cM [mapping data]

Genomic Position: I:1483084..1490474 bp

Function

Mutant Phenotype: Definitions of abbreviations used in the text.

RNAi Phenotype(s): WT [For details see: Ahringer JA 16 Nov 2000]
WT [For details see: Rual JF 01 Sep 2004]

Fig. 3. A Second Sample Table from WormBase.

each *Dimension*. A *Label Sequence* is a sequence of *Label Components* ordered by their *Sequence #*'s. As an example, consider the value, *342 aa*, that appears in Table 7 of Figure 1. Table 7 is two-dimensional, as are many, if not most HTML tables. The first dimension has the label sequence *Identification.Gene model(s).Amino Acids* where the sequence #'s of the label sequence designate identification as the first label component, *Gene model(s)* as the second, and *Amino Acids* as the third. The second dimension has the label sequence *Identification.Gene model(s).1*[2]

[2] 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*.

Although automatic table interpretation can be complex, 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 web site and have similar structures *sibling pages.*[3] 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 web site, 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 ortholog(s).* Further, the tables under *Identification.IDs* also have identical header rows. The data rows, however, vary considerably. General speaking, we can look for commonalities in sibling tables 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 web site and of the individual tables embedded within pages of the web site. With respect to identified labels, we look below or to the right for value associations; we may also need to look above or to the left. In Figure 1, the values for *Identification.Gene model(s).Amino Acids* are below, and the values for *Identification.Speices* are to the right.

In addition to discovering the structure pattern for a web site, we can also dynamically adjust the pattern as we interpret the tables on each retrieved pages. If the system encounters a table that varies from the pattern by having 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 the sibling table of Table 7 in Figure 3 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.

By way of comparison with related work, we note that recent surveys [5, 17] 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. [3, 15]). 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. [2, 6]). These simple assumptions (labels are either the first row or the first column) are easily

[3] Hidden-web pages are usually generated dynamically from a pre-defined templates in response to submitted queries. Therefore hidden-web pages usually have sibling pages.

broken in complex tables. More sophisticated table interpretation techniques have appeared in recent papers [8, 9, 11]. 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.

Other researchers have also tried to take advantage of sibling pages to determine page structure. RoadRunner [4] 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. EXALG [1] uses equivalence classes (sets of items that occur with the same frequency in sibling pages) and differentiating roles to generate extraction templates for the sibling pages. DEPTA [18] compares different records in a page instead of sibling pages and tries to find the extraction template for the record. Our system fundamentally differs from these approaches. These approaches focus on finding data fields. They do 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 tries to find the general structure pattern for the entire web site. It dynamically adjusts the structure pattern as it encounters new, yet-unseen structures.

We call our system *TISP* (*Table Interpretation with Sibling Pages*). We present the details of TISP and our contribution to table interpretation by sibling page comparison in the remainder of the paper as follows. Section 2 provides the details about how TISP analyzes a source page to find tables and match them with tables in sibling pages. Section 3 explains how TISP infers the general structure patterns of a web site and therefore how it interprets the tables from the site. In Section 4, we report the results of experiments we conducted involving sites for car advertisements, molecular biology, and geopolitical information, which we found on the hidden web. In Section 5, we make concluding remarks.

2 Sibling Table Recognition

After obtaining a source document, TISP first parses the source code and locates all HTML components enclosed by <table> and </table> tags (tagged tables). When tagged tables are nested inside of one another, TISP 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. There are also two third level nested tables.

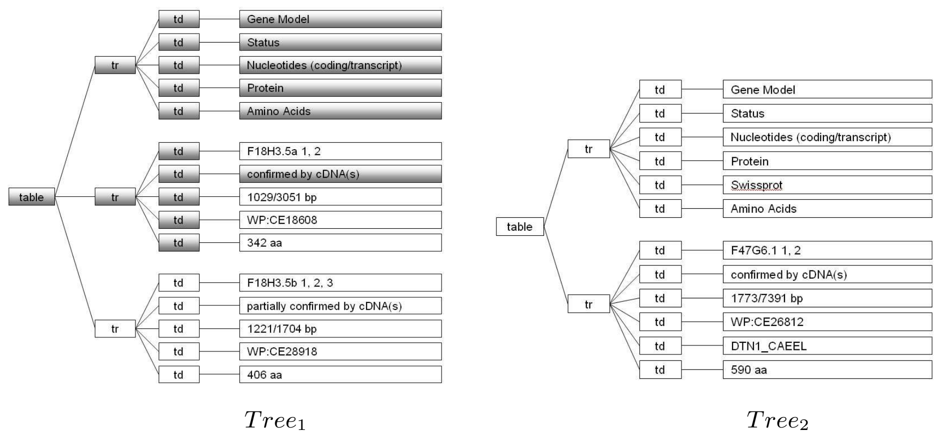


Fig. 4. DOM Trees for Table 7 in Figure 1 and its Sibling Table in Figure 3.

We treat each tagged table as an individual table and assign a *Table ID* to it. If the table is nested, we replace the table in the upper level with its ID number. By so doing, we are able to remove nested tables from upper level tables. As a result, TISP decomposes the page in Figure 1 into a set of tables, each with an ID and a path.

To compare and match tables, we first transform each HTML table into a DOM tree. It is easy to transform our input in Figure 2(a) to a DOM tree, indeed the conceptual-model instance abstractly models a DOM tree for the tables within an HTML page. $Tree_1$ in Figure 4 shows the DOM tree for Table 7 in Figure 1, and $Tree_2$ in Figure 4 shows the DOM tree for its corresponding table in Figure 3.

Tai [12] 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 for all (i_1, j_1), $(i_2, j_2) \in$ M, where i_1 and i_2 are two nodes from T and j_1 and j_2 are two nodes from T':

(1) $i_1 = i_2$ iff $j_1 = j_2$;
(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 [16] 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($(n_1 n_2)$), 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. The highlighted part in $Tree_1$ in Figure 4 shows the matched nodes for $Tree_1$ with respect to $Tree_2$ in Figure 4. The highlighted nodes indicate a match.

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.

We call the maximum number of matched nodes among the two trees 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 [7] 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 4, 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 boostrap selective sampling [13], we first set initial thresholds by empirical observation (90% for the higher threshold and 20% for the lower threshold); then TISP 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. Our example has no false matches. A false match usually happens when a table does not have a corresponding table in the sibling page. In this case, we save the table. When more sibling pages are compared, we might find a matching table.

3 Structure Patterns

The structure pattern of a table tells us how to transform the information contained in the model instances in Figure 2(a) to the model instance in Figure 2(b), and thus how to interpret a table.

Pattern 1:

<table>(<tbody>)?

$\boxed{\text{<tr>}(<(td|th)>\{L\})^n}$

$\boxed{(\text{<tr>}(<(td|th)>\{V\})^n}\,)^+$

Pattern 2:

<table>(<tbody>)?

$(\boxed{< tr >< (td|th) > \{L\}(< (td|th) > \{V\})^n}\,)^+$

Pattern 3:

<table>(<tbody>)?

$\boxed{\text{<tr>}(<(td|th)>\{L\})^n}$

$(\boxed{< tr >< (td|th) > \{L\}(< (td|th) > \{V\})^{(n-1)}}\,)^+$

Fig. 5. Some Basic Pre-defined Pattern Templates.

3.1 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 5). 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 3.4.

Figure 5 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.

3.2 Pattern Generation

To check whether a table matches any pre-defined pattern template, TISP 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. In tables, labels

<table>
<tr> <td>Gene Model <td>Status <td>Nucleotides(coding/transcript)
<td>Protein <td>Amino Acids
(*<tr><td>*$V_{Gene\ Model}$*<td>*V_{Status} *<td>*$V_{Nucleotides(coding/transcript)}$
<td>$V_{Protein}$ *<td>*$V_{Amino\ Acids}$)$^{+}$

Fig. 6. Structure Pattern for Table 7 in Figure 1.

or values are usually grouped. We are seeking for a structure pattern instead of classifying individual cells. Sometimes we find a matched node, but all other nodes in the group are mismatched nodes and agree with a certain pattern (e.g. the highlighted record node in the second subtree in $Tree_1$ in Figure 4.), TISP should ignore the disagreement and assume the mismatched node is a node of value too. Specifically, we calculate a *template match percentage* between a pre-defined pattern 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 boostrap selective sampling, with an initial threshold of 80%.

Consider the mapped result in Figure 4 as an example. The highlighted nodes are matched nodes in $Tree_1$. Comparing the template match percentage for this mapped result for the three pattern templates in Figure 5, 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 $Tree_1$ in Figure 4, 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 5. 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 TISP 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_L for each value. The subscript L for a value V designates the label-value pair for each record in a table. Figure 6 shows the specific structure pattern for Table 7 in Figure 1.

3.3 Pattern Usage

With a structure pattern for a specific table, we can interpret the table and all its sibling tables. The path 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 TISP. (In Section 3.5, we explain how TISP reacts to a failure notification.)

When the pattern matches exactly, TISP can generate the label sequence and value for each label-value pair 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, TISP needs to add another dimension and add row numbers. Since the table is inside of other tables, TISP recursively searches for the tables in the upper levels of nesting and collects all needed labels.

3.4 Pattern Combinations

It is possible that TISP cannot match any pre-defined template. In this case, it looks for pattern combinations. Using Figure 7 as an example, assume that TISP 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 5, 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 columns.

Location	chr8	Strand	+
Sequence Length	5095	Total Exon Length	2161
Number of Exons	4	Number of SNPs	0
Max Exon Length	1044	Min Exon Length	93

Fig. 7. An Example for Pattern Combination from MutDB.

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: $< table >$ $(< tbody >)?$. 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 $< tr >$ tag at the beginning is removed. Figure 8 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 7, where $n = m = 1$, and the plus (+) is 4.

Example 1:
$< table > (< tbody >)?$
$(< tr >< (td|th) >\{\mathrm{L}\}(< (td|th) > \{V\})^n)^+$
$< tr > (< (td|th) > \{L\})^m(< tr > (< (td|th) > \{V\})^m)^+$
Example 2:
$< table > (< tbody >)?$
$(< tr >< (td|th) >\{\mathrm{L}\}(< (td|th) > \{V\})^n< (td|th) >\{\mathrm{L}\}(< (td|th) > \{V\})^m)^+$

Fig. 8. Two Examples of Pattern Combinations.

The initial search for combinations is similar to the search for single patterns. TISP checks patterns until it finds mismatches, it then checks to see whether the mismatched part matches with some other pattern. TISP 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. TISP 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 7, TISP 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 .

3.5 Dynamic Pattern Adjustment

Given a structure pattern for a table, we know where the table is in the source document (its path), the location of the labels and values, and the association between labels and values. When TISP encounters a new sibling page, it tries to locate each sibling table following the path, and then to interpret it by matching it with the sibling table structure pattern. If the new 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 path 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 path, we cannot locate the sibling table for which we are looking. In this case, TISP searches for tables at the same level of nesting, looking for one that matches the pattern. If TISP finds one, it obtains the path and adds it as an alternative. Thus, for future sibling pages, TISP can (in fact, always does) check all alternative paths before searching for another alternative path. If TISP 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

<table>
<tr> <td>Gene Model <td>Status <td>Nucleotides(coding/transcript)
  <td>Protein (<td>Swissprot)? <td>Amino Acids
(*<tr><td>*$V_{\mathit{Gene\ Model}}$*<td>*V_{Status} *<td>*$V_{\mathit{Nucleotides(coding/transcript)}}$
  <td>V_{Protein} (*<td>*$V_{\mathit{Swissprot}}$)? *<td>*$V_{\mathit{Amino\ Acids}}$)$^{+}$

Fig. 9. Structure Pattern for the Table in Figure 3.

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 1) as an example. The table matches the pattern in Figure 6 until we encounter the 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 9 shows.

4 Experimental Results

We tested TISP for three different fields: car advertisements for commercial data, molecular biology for scientific data, and interesting information about U.S. states and about countries for geopolitical data. Most of the source pages were collected from popular and well-known web sites such as cars.com, NCBI database, Wormbase, MTB database, the CIA World Factbook, and the U.S. Geological Survey. We tested more than 2,000 tables found in 275 sibling pages in 35 web sites. For each web site, we randomly chose two sibling pages for initial pattern generation. For the initial two sibling pages, we tested (1) whether TISP was able to recognize HTML data tables and discard HTML tables used only for layout, (2) whether it was able to pair all sibling tables correctly, and (3) 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 (1) whether TISP was able to interpret tables using the recognized structure patterns, (2) whether it correctly detected the need for dynamic adjustment, and (3) whether it 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.[4] TISP correctly discarded all uses of tables for layout and successfully paired all sibling tables. There were no nested tables in this domain. Most of the web sites contained only one table pattern, except for one site that had three different patterns. Two web sites contained tables with structure combinations. TISP successfully interpreted all tables from the generated patterns. No adjustment were needed, neither for any path nor for any label.

[4] The sibling pages in this domain are usually very regular. Indeed, we found no table variations in any of the sites we considered. We, therefore, only tested five pages per site.

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, TISP falsely classified three pairs of layout tables as data tables. TISP, however, successfully eliminated these false sibling pairs during pattern generation because it was unable to find a matching pattern. No false patterns were generated. TISP was able to recognized 28 of 29 structure patterns. TISP missed one pattern because the table contained too many empty cells. If it had considered empty cells as mismatches, TISP would have correctly recognize this pattern. As TISP 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 only partially correctly because TISP considered the irrelevant information *To Top* as a header.

For the geopolitical information domain, we tested 100 sibling pages from 10 different web sites with 884 HTML tables. TISP correctly paired 100% of all data tables and correctly discarded all layout tables. For initial pattern generation, TISP was able to recognize all 22 structure patterns. As TISP 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, TISP 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 (i.e its label components were only partially correct).

For measuring the overall accuracy of TISP, we computed precision (P), recall (R), and an F-measure ($F = 2PR/(P{+}R)$). In its table recognition step, TISP correctly discarded 155 of 158 layout tables and discarded no data tables. It therefore achieved an F-measure of 99.0% (98.1% recall and 100% precision). TISP later discarded these three layout tables in its pattern generation step, but it also rejected two data tables, being unable to find any pattern for them. It thus achieved an F-measure of 99.4% (100% recall and 98.8% precision). For table interpretation, TISP correctly recognized 69 of 74 structure patterns. It therefore achieved a recall of 93.2%. Of the 72 structure patterns it detected, 69 were correct. It therefore achieved a precision of 95.8%. Overall the F-measure for table interpretation was 94.5% for the sites we tested.

We discuss the time performance of TISP in two phases: (1) initial pattern generation from a pair of sibling pages and (2) interpretation of the tables in the rest of the sibling pages. The time for pattern generation given a pair of sibling pages consists of: (1) the time to read and parse the two pages and locate all the HTML tables, (2) the time for sibling table comparisons, and (3) the time to select from pre-defined structure templates and generate a pattern. The complexity of parsing and locating HTML tables is $O(n)$, where n is the number of HTML tags. The simple tree matching algorithm has time complexity $O(m_1 m_2)$, where m_1 and m_2 are the numbers of nodes of the two sibling trees. To find the best match for each HTML table, we need to compare each table with all the HTML tables in its sibling page. The time complexity is $O(km_1m_2)$, where k is the number of HTML tables in the sibling page. The time complexity for finding the correct pattern for each matched sibling table is $O(pl)$, where p is the number of pattern templates and l is the number of leaf nodes in the HTML table. If pattern combinations are involved, the complexity of template discovery increases multiplicatively since for each subgroup we must consider every template and find the best match. We conducted the experiment on a Pentium 4 computer running at 3.2 GHz. The typical actual time needed for the pattern generation for a pair of sibling pages was below or about one second. The actual time reached a maximum of 15 seconds for a complicated web site where pages had more than 20 tables.

The time for table interpretation for a single sibling web page when no adjustment is necessary consists of: (1) the time for locating each table and (2) the time for processing the table with a pattern. The complexity of locating a table is $O(p)$, where p is the number of path possibilities leading to the table. Each path possibility is itself logarithmic with respect to the number of nodes in the DOM tree for the pages. The complexity of matching a located table with the corresponding pattern is $O(el)$, where e is the number of pattern entries (an entry could be either a pattern label or a pattern value) of the pattern and l is the number of leaf nodes in the HTML table's DOM tree. The time to do adjustments ranges from the time to do a simple label adjustment, which is constant, to the time required to re-evaluate all sibling tables, which is the same as the time for initial pattern generation. Overall, the typical actual time needed for interpreting tables in one page was below one second. The actual time reached a maximum of 19 seconds for a complicated web page with several tables and several adjustments.

5 Concluding Remarks

In this paper we introduced TISP, which provides a way to automatically interpret tables in hidden-web pages—pages which are almost always sibling pages. By comparing data tables in sibling pages, TISP is able to find the location of table labels and data entries, and pair them to infer the general pattern for all sibling tables from the same site. Our experiments using source pages from three different domains—car advertisements, molecular biology, and geopolitical information—indicate that TISP can succeed in properly interpreting tables in sibling pages. TISP achieved an F-measure for sibling table interpretation of 94.5%.

References

1. A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD'03)*, pages 337–348, San Diego, CA, 2003.
2. H. Chen, S. Tsai, and J. Tsai. Mining tables from large scale HTML texts. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING'00)*, pages 166–172, Saarbrcken, German, July–August 2000.

3. W. W. Cohen, M. Hurst, and L.S. Jensen. A flexible learning system for wrapping tables and lists in HTML documents. In *Proceedings of the International World Wide Web Conference (WWW'02)*, pages 232–241, Honolulu, Hawaii, May 2002.
4. V. Crescenzi, G. Mecca, and P. Merialdo. RoadRunner: Towards automatic data extraction from large web sites. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB'01)*, pages 109–118, Rome, Italy, September 2001.
5. D.W. Embley, M. Hurst, D. Lopresti, and G. Nagy. Table processing paradigms: A research survey. *International Journal of Document Analysis and Recognition*, 8(2-3):66–86, June 2006.
6. D.W. Embley, C. Tao, and S.W. Liddle. Automating the extraction of data from HTML tables with unknown structure. *Data & Knowledge Engineering*, 54(1):3–28, July 2005.
7. D. Gale and L.S. Shapley. College admissions and the stability of marriage. *American Mathematics Monthly*, 69:9–14, 1962.
8. W. Gatterbauer and P. Bohunsky. Table extraction using spatial reasoning on the CSS2 visual box model. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI'06)*, pages 1313–1318, Boston, Massachusetts, July 2006.
9. W. Gatterbauer, P. Bohunsky, M. Herzog, B. Krupl, and B. Pollak. Towards domain-independent information extraction from web tables. In *Proceedings of the 16th International World Wide Web Conference (WWW'07)*, Banff, Canada, May 2007. (in press).
10. P.G. Ipeirotis, L. Gravano, and M. Sahami. Probe, count, and classify: categorizing hidden web databases. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data (SIGMOD'01)*, pages 67–78, Santa Barbara, California, May 2001.
11. A. Pivk, P. Cimiano, and Y. Sure. From tables to frames. In *Proceedings of the Third International Semantic Web Conference (ISWC'04)*, pages 166–181, Hiroshima, Japan, November 2004.
12. K.-C. Tai. The tree-to-tree correction problem. *Journal of the ACM*, 26(3):422–433, 1979.
13. C.A. Thompson, M.E. Califf, and R.J. Mooney. Active learning for natural language parsing and information extraction. In *Proceedings of 16th International Conference on Machine Learning*, pages 406–414, Bled, Slovenia, June 1999.
14. X. Wang. *Tabular Abstraction, Editing, and Formatting.* PhD thesis, Univeristy of Waterloo, 1996.
15. Y. Wang and J. Hu. A machine learning based approach for table detection on the web. In *Proceedings of the 11th International Conference on World Wide Web (WWW'02)*, pages 242–250, Honolulu, Hawaii, May 2002.
16. W. Yang. Identifying syntactic differences between two programs. *Software Practice and Experience*, 21(7):739–755, 1991.
17. R. Zanibbi, D. Blostein, and J.R. Cordy. A survey of table recognition. *International Journal of Document Analysis and Recognition*, 7(1):1–16, 2004.
18. Y. Zhai and B. Liu. Web data extraction based on partial tree alignment. In *Proceedings of the 14th International Conference on World Wide Web (WWW'05)*, pages 76–85, Chiba, Japan, May 2005.