Data & Knowledge Engineering 68 (2009) 683–703

Contents lists available at ScienceDirect

Data & Knowledge Engineering

journal homepage: www.elsevier.com/locate/datak

Automatic hidden-web table interpretation, conceptualization, and semantic annotation

Cui Tao *, David W. Embley

Department of Computer Science, Brigham Young University, Provo, UT 84602, USA

ARTICLE INFO

Article history:
Available online 1 March 2009

Keywords:
Automatic table interpretation
Ontology generation
Automatic semantic annotation
Web of data

ABSTRACT

The longstanding problem of automatic table interpretation still eludes 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, semantic annotation, and semi-structured data management. In this paper, we offer a 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. Our system compares them to identify and connect nonvarying components (category labels) and varying components (data values). We tested our solution using more than 2000 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 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%. Further, given that we can automatically interpret tables, we next show that this leads immediately to a conceptualization of the data in these interpreted tables and thus also to a way to semantically annotate these interpreted tables with respect to the ontological conceptualization. Labels in nested table structures yield ontological concepts and interrelationships among these concepts, and associated data values become annotated information. We further show that semantically annotated data leads immediately to queriable data. Thus, the entire process, which is fully automatic, transform facts embedded within tables into facts accessible by standard query engines.

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 these hidden-web pages and making their facts externally accessible 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. Once interpreted, we can automatically annotate the information in these pages and make it available to standard query engines.

* Corresponding author.
E-mail address: ctao@cs.byu.edu (C. Tao).

[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 [23].

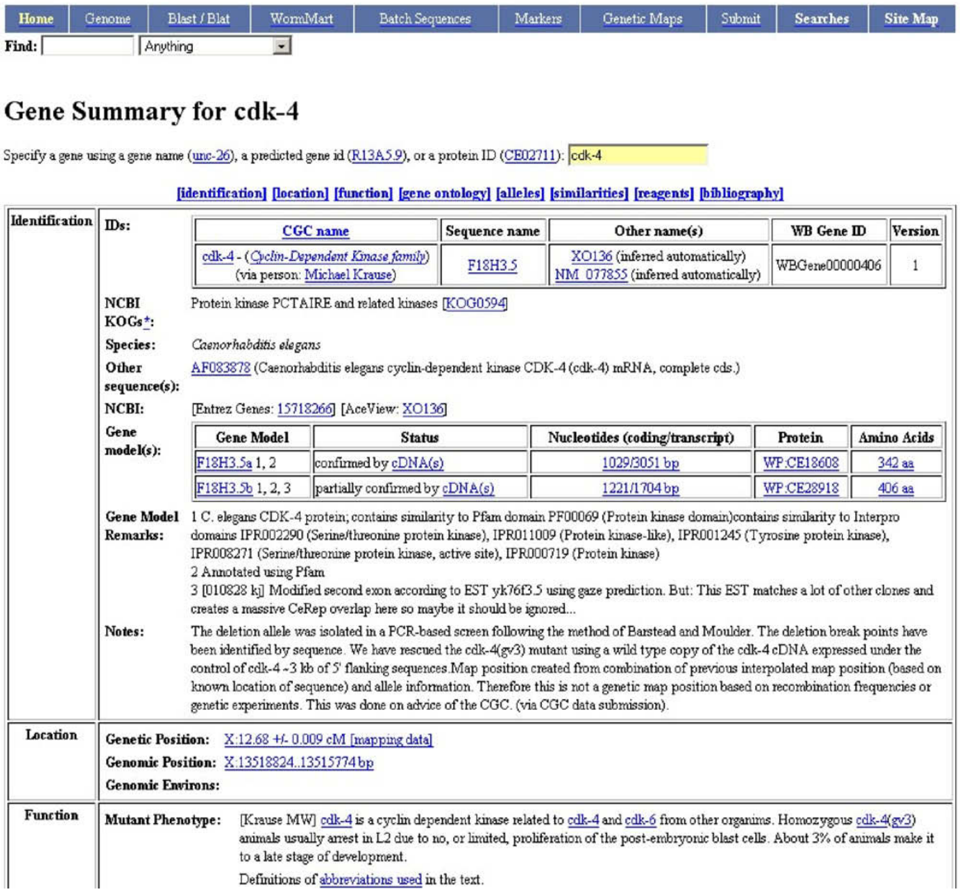
0169-023X/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.datak.2009.02.010

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. Fig. 1 shows an example. Tables may be nested or conjoined as are the tables in Fig. 1. Labels may span across several cells to give a general description as does *Identification* and *Location* in Fig. 1. 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.

To interpret a table is to properly associate table category labels with table data values. Using Fig. 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, Sequence 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 Fig. 1, we match category labels with values as Fig. 2 shows. We associate one or more sequences of labels with each data value in a table. Borrowing notation from Wang [41], 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 two label-value pairs in Fig. 2, there is only one label sequence. The third, however, has two: *Identification.Gene model(s).Amino Acids* and *2*. 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 records with sequence numbers—the first record 1 and the second record 2.



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): cdk-4

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

Identification

IDs:

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

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

Gene Model Remarks: 1 C. elegans CDK-4 protein; contains similarity to Pfam domain PF00069 (Protein kinase domain)contains similarity to Interpro domains IPR002290 (Serine/threonine protein kinase), IPR011009 (Protein kinase-like), IPR001245 (Tyrosine protein kinase), IPR008271 (Serine/threonine protein kinase, active site), IPR000719 (Protein kinase)
2 Annotated using Pfam
3 [010828 kj] Modified second exon according to EST yk76f3.5 using gaze prediction. But: This EST matches a lot of other clones and creates a massive CeRep overlap here so maybe it should be ignored...

Notes: The deletion allele was isolated in a PCR-based screen following the method of Barstead and Moulder. The deletion break points have been identified by sequence. We have rescued the cdk-4(gv3) mutant using a wild type copy of the cdk-4 cDNA expressed under the control of cdk-4 ~3 kb of 5' flanking sequences.Map position created from combination of previous interpolated map position (based on known location of sequence) and allele information. Therefore this is not a genetic map position based on recombination frequencies or genetic experiments. This was done on advice of the CGC. (via CGC data submission).

Location

Genetic Position: X:12.68 +/- 0.009 cM [mapping data]
Genomic Position: X:13518824..13515774 bp
Genomic Environs:

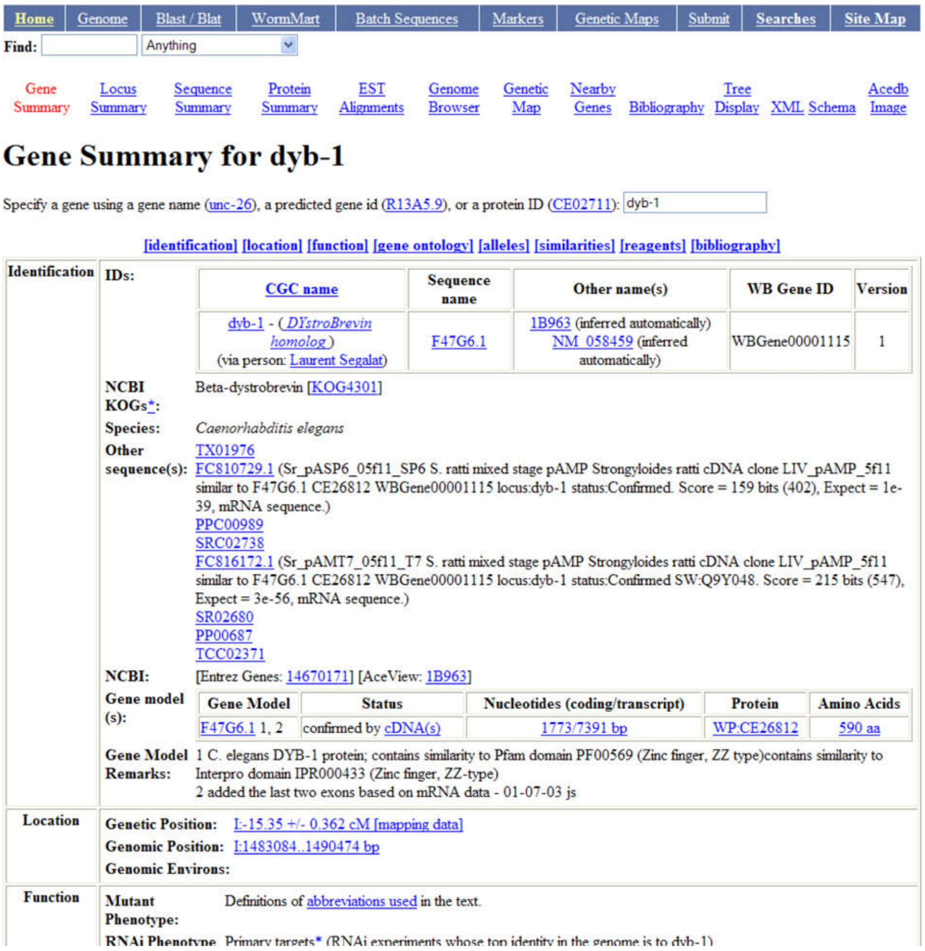
Function

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.

Fig. 1. A sample table from WormBase [44].

$(Identification.IDs.CGC\ name) \mapsto$
$\quad cdk\text{-}4\text{-}(Cyclin\text{-}Dependent\ Kinase\ family)$
$\quad (via\ person{:}\ Michael\ Krause);$
$(Identification.IDs.Sequence\ name) \mapsto F18H3.5;$
…
$(Identification.Gene\ model(s).Amino\ Acids,\ 2) \mapsto 406\ aa;$
…

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


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

Find: Anything

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] [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	1B963 (inferred automatically) NM_058459 (inferred automatically)	WBGene00001115	1

NCBI KOGs*: Beta-dystrobrevin [KOG4301]

Species: *Caenorhabditis elegans*

Other sequence(s): TX01976
FC810729.1 (Sr_pASP6_05f11_SP6 S. ratti mixed stage pAMP Strongyloides ratti cDNA clone LIV_pAMP_5f11 similar to F47G6.1 CE26812 WBGene00001115 locus:dyb-1 status:Confirmed. Score = 159 bits (402), Expect = 1e-39, mRNA sequence.)
PPC00989
SRC02738
FC816172.1 (Sr_pAMT7_05f11_T7 S. ratti mixed stage pAMP Strongyloides ratti cDNA clone LIV_pAMP_5f11 similar to F47G6.1 CE26812 WBGene00001115 locus:dyb-1 status:Confirmed SW:Q9Y048. Score = 215 bits (547), Expect = 3e-56, mRNA sequence.)
SR02680
PP00687
TCC02371

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

Gene model (s):

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

Gene Model Remarks: 1 C. elegans DYB-1 protein; contains similarity to Pfam domain PF00569 (Zinc finger, ZZ type)contains similarity to Interpro domain IPR000433 (Zinc finger, ZZ-type)
2 added the last two exons based on mRNA data - 01-07-03 js

Location

Genetic Position: I:-15.35 +/- 0.362 cM [mapping data]
Genomic Position: I:1483084..1490474 bp
Genomic Environs:

Function

Mutant Phenotype: Definitions of abbreviations used in the text.
RNAi Phenotype Primary targets* (RNAi experiments whose top identity in the genome is to dyb-1)

Fig. 3. A second sample table from WormBase.

Thus, the label-value association becomes $(Identification.Genemodel(s).AminoAcids, 2) \mapsto 406\ aa$ where *Identification.Gene model(s).Amino Acids* is the label for the first dimension, and *2* is the row label for the second dimension.

Although automatic table interpretation can be complex, if we have another page, such as the one in Fig. 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*.[2] The two pages in Figs. 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

[2] Hidden-web pages are usually generated dynamically from a pre-defined templates in response to submitted queries, therefore they are usually sibling pages.

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 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.[3] 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 Fig. 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 about being too strict. Sometimes there are additional or missing label-value pairs. The two nested tables whose first column header is *Gene Model* in Figs. 1 and 3 do not share exactly the same structure. The table in Fig. 1 has five columns and three rows, while the table in Fig. 3 has six columns and two rows. Although they have these differences, 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 Fig. 3 only has an additional *Swissprot* column inserted between the *Protein* and *Amino Acids* columns. It is still not difficult, however, to tell that the top rows are rows for labels.

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.

We call our table-interpretation system *TISP* (*Table Interpretation with Sibling Pages*) [36]. Given that we can interpret a table, we can immediately conceptualize it and add semantic annotation to it. We augmented TISP by adding two new functions: conceptualization and annotation. We call the new system TISP++. Given a structure pattern, TISP++ can automatically generate an OWL ontology that conceptualizes the pattern. TISP++ uses an OWL class to represent a table, an OWL object property to represent the nesting between two tables, and an OWL data type property to represent a label. The ontology also declares constraints such as optional and functional if applicable. After the OWL ontology is generated, TISP++ can also automatically annotate all the sibling pages with respect to this ontology. The annotated information is available to any SPARQL query platform, so that users can query and locate information of interest. By doing so, TISP++ makes hidden-web information visible to users from outside specialized hand-built GUIs.

We present the details of TISP and our contribution to table interpretation by sibling page comparison, and the details of TISP++ and our contribution to ontology generation and semantic annotation in the remainder of the paper as follows. Section 2 provides the details about how TISP 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 and find data tables. Section 5 explains how TISP infers the general structure patterns of a web site and therefore how it interprets the tables from the site. Section 5 also explains how to automatically adjust the generated patterns when variations are encountered. Since in Sections 2–5, we present our algorithm somewhat informally, using more exposition and examples than formalism, we also provide in an Appendix a complete and more formal description of the entire process. 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 hidden web. Section 7 explains how TISP++ generates an OWL ontology depending on a structure pattern, and Section 8 explains how TISP++ annotates information automatically. Section 9 discusses related work. In Section 10, we draw conclusions and mention some possibilities for future work.

2. Initial table processing

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, TISP 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, and proceed with our discussion of HTML tables.

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 Fig. 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.

[3] "Structure patterns" are the pattern expressions (path expressions and regular expressions) we use to identify the location of tables within an HTML page and to associate table labels with table values.

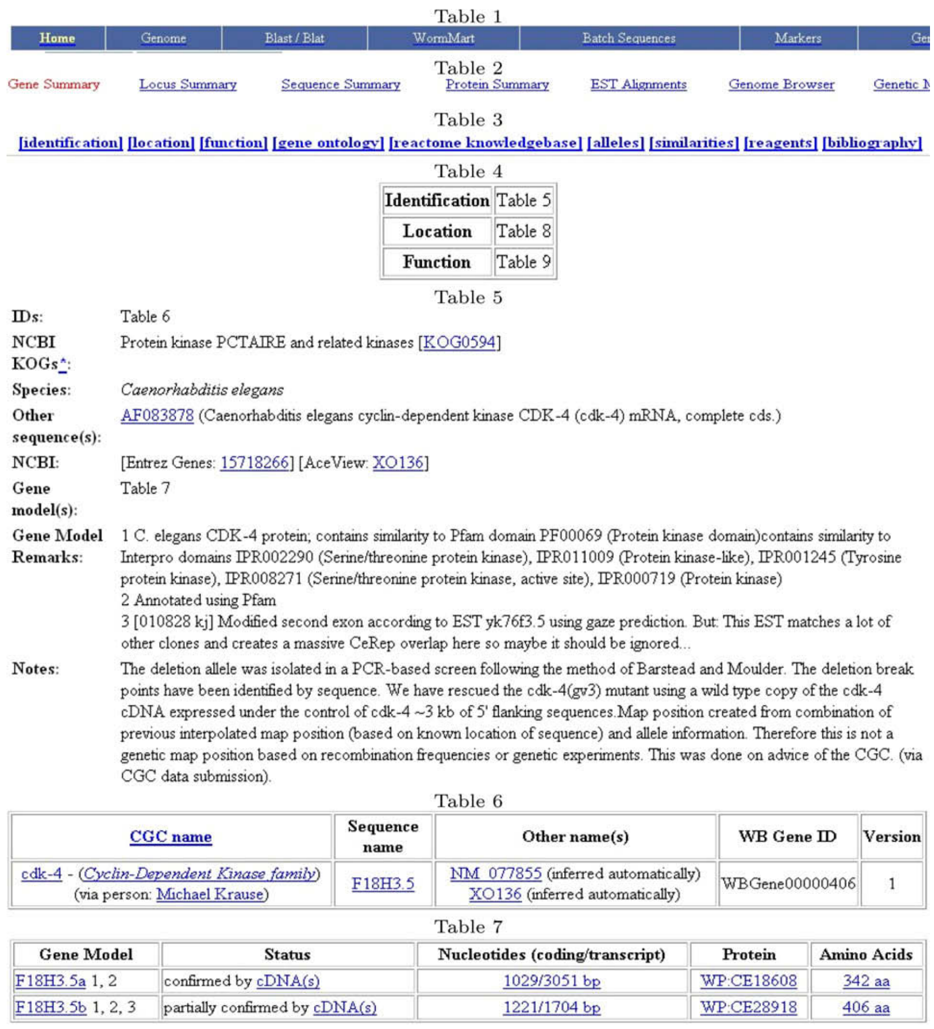


Table 1

Home	Genome	Blast / Blat	WormMart	Batch Sequences	Markers	Ge

Table 2

Gene Summary	Locus Summary	Sequence Summary	Protein Summary	EST Alignments	Genome Browser	Genetic N

Table 3

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

Table 4

Identification	Table 5
Location	Table 8
Function	Table 9

Table 5

IDs:	Table 6
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):	Table 7
Gene Model Remarks:	1 C. elegans CDK-4 protein; contains similarity to Pfam domain PF00069 (Protein kinase domain)contains similarity to Interpro domains IPR002290 (Serine/threonine protein kinase), IPR011009 (Protein kinase-like), IPR001245 (Tyrosine protein kinase), IPR008271 (Serine/threonine protein kinase, active site), IPR000719 (Protein kinase) 2 Annotated using Pfam 3 [010828 kj] Modified second exon according to EST yk76f3.5 using gaze prediction. But: This EST matches a lot of other clones and creates a massive CeRep overlap here so maybe it should be ignored...
Notes:	The deletion allele was isolated in a PCR-based screen following the method of Barstead and Moulder. The deletion break points have been identified by sequence. We have rescued the cdk-4(gv3) mutant using a wild type copy of the cdk-4 cDNA expressed under the control of cdk-4 ~3 kb of 5' flanking sequences.Map position created from combination of previous interpolated map position (based on known location of sequence) and allele information. Therefore this is not a genetic map position based on recombination frequencies or genetic experiments. This was done on advice of the CGC. (via CGC data submission).

Table 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

Table 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

Fig. 4. Decomposition for the tables in Fig. 1.

Fig. 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 Fig. 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.

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, TISP decomposes the page in Fig. 1 into the set of tables in Fig. 4.

3. Table matching

To compare and match tables, we first transform each HTML table into a DOM tree [13]. *Tree*$_1$ in Fig. 5 shows the DOM tree for Table 7 in Fig. 4, and *Tree*$_2$ in Fig. 5 shows the DOM tree for its corresponding table in Fig. 3.

Tai [35] 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 ith 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 $T'[j_2]$.

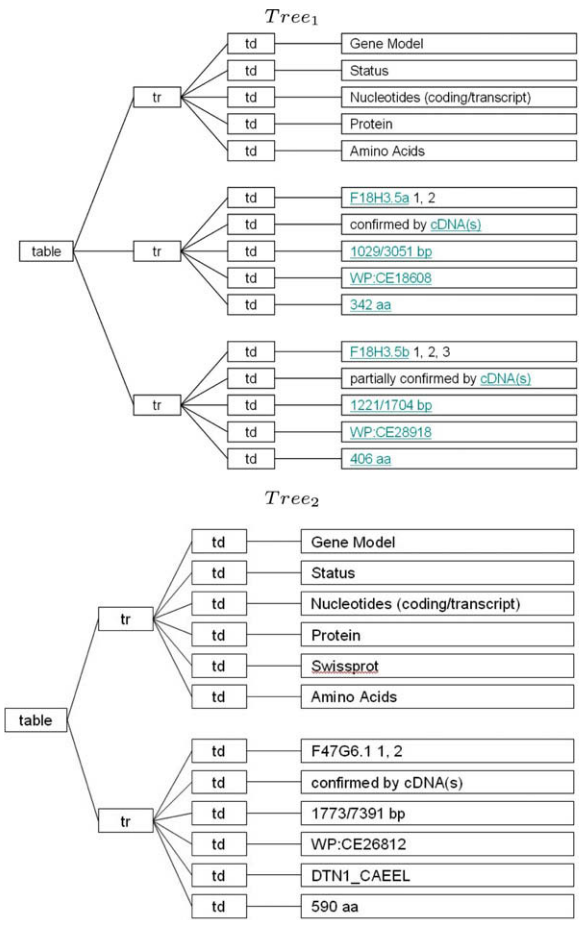


Fig. 5. DOM trees for Table 7 in Fig. 4 and its sibling table in Fig. 3.

According to this definition, each node appears at most once in a mapping—the order between sibling nodes and the hierarchical relation between nodes being 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 [46] 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 Fig. 5 shows the matched nodes for $Tree_1$ with respect to $tree_2$ in Fig. 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. Sibling tables should have a one-to-one correspondence. Based on the match scores, we use the Gale–Shapley stable marriage algorithm [18] to pair sibling tables one-to-one from two sibling pages.

For each pair of 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 Fig. 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 table matches 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 table matches: 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. After some observation, we set the upper threshold at 90% and the lower threshold at 20%.

In our example, Tables 1–3 have match percentages of 100% with their sibling tables. The match percentages for Tables 4–7, 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.

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 [45], which describes the path of the table from the root HTML tag of the document. For example, The location for Table 7 in Fig. 4 is: /html/table[4]/tbody/tr[1]/td[2]/table[2]/tbody/tr[1]/td[2]. An XPath simply lists the nodes (HTML tag names) of a path in a 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 TISP uses the generated pattern to extract label-value pairs from the table and how TISP produces 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 TISP dynamically adjusts a pattern to accommodate table variations it may encounter as it extracts label-value pairs from sibling tables in the web site (Section 5.5).

5.1. Pattern templates

We use regular expressions 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 Fig. 6). 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 repetition symbol to designate a specific number of repetitions. 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.

Fig. 6 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. As future work, we could define additional patterns and experiment with them or allow users to define additional patterns, but the patterns and combinations of patterns we have constitute a large majority of HTML tables.

5.2. 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 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 mis-matched nodes and agree with a certain pattern, TISP should ignore the disagreement and assume the matched node is a mis-matched node of values 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

Pattern 1:

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

Pattern 2:

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

Pattern 3:

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

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

Fig. 6. Some basic pre-defined pattern templates.

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, which we set at 80%.

Consider the mapped result in Fig. 5 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 Fig. 6, 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 $tree_1$ in Fig. 5, 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 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 for the label-value-pair for each record in a table. Fig. 7 shows the specific structure pattern for Table 7 in Fig. 4.

5.3. Pattern usage

With a structure pattern for a specific table, we can interpret the table and all 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 TISP. (In Section 5.5, we explain how TISP reacts to a failure notification.)

When the pattern matches exactly, TISP can generate an interpretation for the table. For our example, the chosen pattern is Pattern 1 (a table with column headers and one or more data rows). 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.

5.4. Pattern combinations

It is possible that TISP cannot match any pre-defined template. In this case, it looks for pattern combinations. Using Fig. 8 as an example, assume that TISP matches 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 Fig. 6, 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.

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. Fig. 9 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 Fig. 8, where $n = m = 1$, and the plus (+) is 4.

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 mis-matched 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 or the process fails to finding any matching template.

/html/table[4]/tbody/tr[1]/td[2]/table[2]/tbody/tr[1]/td[2]
$< table >< tr >$
$< td >$ $Gene\ Model$
$< td >$ $Status$
$< td >$ $Nucleotides(coding/transcript)$
$< td >$ $Protein$
$< td >$ $Amino\ Acids$
$(< tr >$
$< td > V_{Gene\ Model}$
$< td > V_{Status}$
$< td > V_{Nucleotides(coding/transcript)}$
$< td > V_{Protein}$
$< td > V_{Amino\ Acids})^{+}$

Fig. 7. Structure pattern for Table 7 in Fig. 4.

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. 8. An example for pattern combination from MutDB.

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. 9. Two examples of pattern combinations.

For the example in Fig. 8, 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 Pattern 2 used twice.

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 TISP encounters a new sibling page, it tries to locate each sibling table following the XPath, and then tries to interpret it by matching it with the sibling-table structure pattern. If the 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, TISP searches for tables at the same level of nesting, looking for one that matches the pattern. If TISP finds one, it obtains the XPath and adds it as an alternative. Thus, for future sibling pages, TISP can (in fact, always does) check all alternative XPaths before searching for another alternative XPath. 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 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 Fig. 3 (the sibling table of Table 7 in Fig. 4) as an example. The table matches the pattern in Fig. 7 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 Fig. 10 shows.

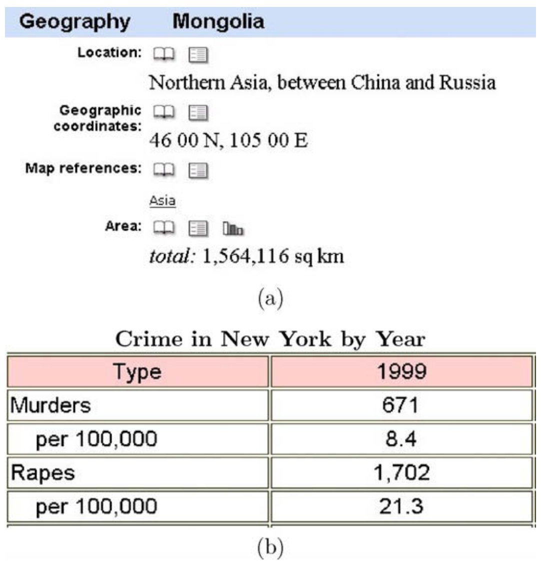
6. Experimental results

We tested TISP using source pages from commercial data, scientific data, and geopolitical data. We picked pages from each field: car advertisements for commercial data, molecular biology for scientific data, and interesting information about US 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 (Mouse Tumor Biology Database), CIA's World Factbook, and US Geological Survey (usgs.gov). We tested more than 2000 tables found in 275 sibling pages in 35 web sites. Most pages from the molecular biology domain and the geopolitical domain have relatively complicated structures. Seven out of 10 sites in the geopolitical domain and eight out of ten sites in the molecular biology domain contained multiple data tables per page. Two of the geopolitical sites and eight of the molecular biology sites contained nested HTML tables.

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

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

Fig. 10. Structure pattern for the table in Fig. 3.



Geography	Mongolia
Location:	Northern Asia, between China and Russia
Geographic coordinates:	46 00 N, 105 00 E
Map references:	Asia
Area:	*total:* 1,564,116 sq km

(a)

Crime in New York by Year

Type	1999
Murders	671
per 100,000	8.4
Rapes	1,702
per 100,000	21.3

(b)

Fig. 11. Two partially misinterpreted tables.

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. Of the 17 pairs of sibling tables, TISP recognized 16 correctly. The one pattern TISP failed to recognize correctly contained too many value cells that included the same value (values such as *unknown*, *general car*, *auto*, *dealer*, and empty spaces). TISP considered them as labels, and thus could not match the table with any pre-defined pattern template or detect any pattern combination. TISP 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. 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 successfully. 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 these tables only partially correctly. Fig. 11 shows these two cases. There are actually two HTML tables in Fig. 11a. The header *Geography Mongolia* is in one HTML table, and the rest of information is in another HTML table. Because it separated tables using HTML tags, TISP was not able to determine the relationship between these two HTML tables. TISP correctly interpreted Fig. 11b as Pattern 3. It, however, did not recognize 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, 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 recognize 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 recognized 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 five 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 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

[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.

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 the 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 $O(m_1m_2)$, 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 exponentially 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 1 s. The actual time reached a maximum of 15 s 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 s for a complicated web page with several tables and several adjustments.

7. Semantic ontology generation

After we obtain the structure pattern for a web site, we can conceptualize the information present in the tables by generating an ontology according to the structure pattern. TISP++ uses a structure pattern to generate OWL classes, properties, and constraints according to the structure pattern. It then uses Jena [24], a semantic web framework for Java, to output the OWL ontology.

Fig. 12 shows part of the ontology for the sibling pages processed from the WormBase gene repository [44]. As a default, TISP++ selects the site name, "WormBase", but a user can change the name if desired. This name provides an anchor to which we attach ontological concepts. Line 10 in Fig. 12 shows the OWL class "WormBase".

For each table label, TISP++ generates an OWL class. The label name becomes the class name. To satisfy the OWL syntax, however, TISP++ elides illegal characters such as spaces and parentheses. Thus "Gene model(s):" becomes "Genemodels" as Line 13 in Fig. 12 shows. The generated ontology also represents the relationships among the labels. TISP++ generates relationships according to the structure patterns in Fig. 6. For Pattern 1 and Pattern 2, each value has only one associated label, and each label has only one parent label. Thus, these patterns require only binary relationships and relationship generation is straightforward. For a binary relationship between two classes *A* and *B*, TISP++ generates an OWL object property: *A–B* and its inverse *B–A*. For the property *A–B*, TISP++ defines *A* as the domain and *B* as the range. For example, Lines 17–23 in Fig. 12 show the OWL object property for *WormBase-Identification*. If a label is paired with an actual value such as are the labels *Gene Model* and *Amino Acids* in Figs. 1 and 3, TISP++ generates an OWL data type property for the OWL class associated with this label. For example, data type property *AminoAcidsValue* describes the actual value for *AminoAcids*. As Lines 31–34 in Fig. 12 show, its domain is *AminoAcids* and its range is string, by default.

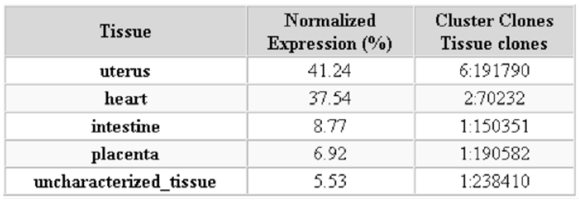
For Pattern 3, each value has two associated labels. Fig. 13 shows an example: the value *41.24* has the labels *Normalized Expression* (%) and *uterus*. Thus, the pattern requires a ternary relationship. Since OWL ontologies only allow binary relationships, we transform Pattern 3 as follows. In Fig. 6, we consider the label *L* in the pattern "$(<tr><(td|th)>\{L\}(<(td|th)>\{V\})^{(n-1)})^{+}$" as a value *V*. The pattern then becomes "$(<tr><(td|th)>\{V\}(<(td|th)>\{V\})^{(n-1)})^{+}$" and can be further simplified to "$(<tr>(<(td|th)>\{V\})^{n})^{+}$", which is the same as Pattern 1. In Fig. 13, for example, the contents in the first row and the first column are all labels. When TISP++ generates the ontology for this table, it considers the contents in the first row as labels, but considers the contents in the first column as values (except for the first one "Tissue" since it is in the first row). Therefore TISP++ can transform ternary relationships to binary relationships and then translate them as binary relationships to OWL.

8. Semantic annotation and querying

After TISP++ generates an ontology according to the structure pattern of a web repository, it automatically annotates the pages from this repository with respect to the generated ontology. Fig. 14 shows a portion of the RDF file generated from the

```
<rdf:RDF
        xmlns:owl="http://www.w3.org/2002/07/owl#"
        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
        xmlns:base ="http://dithers.cs.byu.edu/owl/ontologies/wormbase#"... >
...
    <owl:Ontology rdf:about="">
        <rdfs:comment>OWL Ontology for WormBase</rdfs:comment>
        <rdfs:label>WormBase Ontology</rdfs:label>
    </owl:Ontology>
    <owl:Class rdf:ID="WormBase"/>
    <owl:Class rdf:ID="Identification"/>
    ...
    <owl:Class rdf:ID="Genemodels"/>
    ...
    <owl:Class rdf:ID="AminoAcids"/>
    ...
    <owl:ObjectProperty rdf:ID="WormBase-Identification">
        <owl:inverseOf>
            <owl:ObjectProperty rdf:ID="Identification-WormBase">
        </owl:inverseOf>
        <rdfs:domain rdf:resource="#WormBase"/>
        <rdfs:range rdf:resource="#Identification"/>
    </owl:ObjectProperty>
    ...
    <owl:ObjectProperty rdf:ID="Identification-IDs">
    ...
    <owl:ObjectProperty rdf:ID="Identification-Genemodels">
    ...
    <owl:ObjectProperty rdf:ID="Genemodels-AminoAcids">
...
    <owl:DatatypeProperty rdf:ID="AminoAcidsValue">
        <rdfs:range rdf:resource="xsd;string"/>
        <rdfs:domain rdf:resource="#AminoAcids"/>
    </owl:DatatypeProperty>
...
</rdf:RDF>
```

Fig. 12. Partial OWL ontology for WormBase.



Tissue	Normalized Expression (%)	Cluster Clones Tissue clones
uterus	41.24	6:191790
heart	37.54	2:70232
intestine	8.77	1:150351
placenta	6.92	1:190582
uncharacterized_tissue	5.53	1:238410

Fig. 13. An example table with Pattern 3.

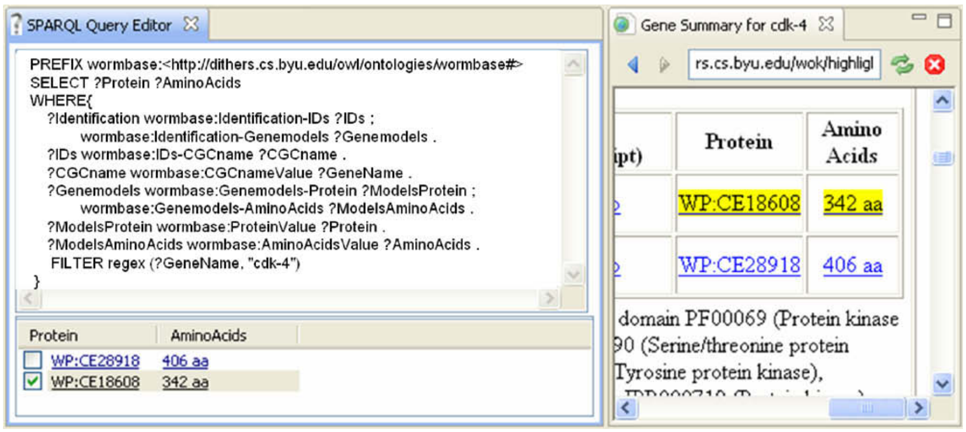
page in Fig. 1. In an RDF annotation file, we first declare the name spaces of referenced ontologies. Line 2 in Fig. 14 refers to the *wormbase* ontology in Fig. 12, and Lines 3 and 4 declare additional name spaces. The *annotation* name space gives the meaning of the annotation declarations in the RDF file, and *22-rdf-syntax-ns* name space gives the meaning of the rdf declarations.

In Line 6, the URI instance *WormBase_1*, which refers to the whole table, is a URI instance for the *WormBase* class in the *wormbase* ontology in Fig. 12. In Line 11, the URI instance *Identification_1*, which refers to the value of the label *Identification* in Table 4 in Fig. 4, is a URI instance of the class *Identification* in the *wormbase* ontology. Since Table 7 in Fig. 4 has two data rows, TISP++ declares URI instances: *Genemodels_1* and *Genemodels_2*, one for each row (Lines 14 and 15). TISP++ also declares the relationship between two URI instances. For example, TISP++ declares the relationship *WormBase_1-Identification_1* in Lines 6 and 7. TISP++ also declares the relationships between instances and annotated values. Annotated values appear as themselves, tagged with both *HTMLText* and with their appropriate *<label>Value* (for string types the HTML text and the value are often the same. But for data types like *xsd:boolean*, the HTML text could be "True" or "False", but its value in the value space should be "T" or "F"). For example, Line 44 gives the *HTMLText* for the protein value "WP:CE18608", and

```
1.  <rdf:RDF
2.          xmlns:wormbase=“http://www.deg.byu.edu/owl/ontologies/wormbase#”
3.          xmlns:ann=“http://www.deg.byu.edu/owl/ontologies/annotation#”
4.          xmlns:rdf=“http://www.w3.org/1999/02/22-rdf-syntax-ns#”>
5.      …
6.      <wormbase:WormBase rdf:ID=“WormBase 1”>
7.          <wormbase:WormBase-Identification rdf:resource=”#Identification 1”/>
8.          <wormbase:WormBase-Location rdf:resource=”#Location 1”/>
9.          …
10.     </wormbase:WormBase>
11.     <wormbase:Identification rdf:ID=“Identification 1”>
12.         <wormbase:Identification-IDs rdf:resource=“#IDs 1”/>
13.         …
14.         <wormbase:Identification-Genemodels rdf:resource=“#Genemodels 1”/>
15.         <wormbase:Identification-Genemodels rdf:resource=“#Genemodels 2”/>
16.         …
17.     </wormbase:Identification>
18.     <wormbase:IDs rdf:ID=“IDs 1”>
19.         <wormbase:IDs-CGCname rdf:resource=“#CGCname 1”/>
20.         <wormbase:IDs-Sequencename rdf:resource=“#Sequencename 1”/>
21.         …
22.     </wormbase:IDs>
23.     …
24.     <wormbase:Genemodels rdf:ID=“Genemodels 1”>
25.         …
26.         <wormbase:Genemodels-Protein rdf:resource=“#Protein 1”/>
27.         <wormbase:Genemodels-AminoAcids rdf:resource=“#AminoAcids 1”/>
28.     </wormbase:Genemodels>
29.     <wormbase:Genemodels rdf:ID=“Genemodels 2”>
30.         …
31.         <wormbase:Genemodels-Protein rdf:resource=“#Protein 2”/>
32.         <wormbase:Genemodels-AminoAcids rdf:resource=“#AminoAcids 2”/>
33.     </wormbase:Genemodels>
34.     …
35.     <wormbase:CGCname rdf:ID=“CGCname 1”>
36.         <ann:OffsetOnHTMLPage/>5951< /ann:OffsetOnHTMLPage/>
37.         <ann:HTMLText/>cdk-4 … < /ann:HTMLText/>
38.         <wormbase:CGCnameValue>cdk-4 … < /wormbase:ProteinValue>
39.         <wormbase:CGCname-IDs rdf:resource=”#IDs 1”/>
40.     </wormbase:CGCname>
41.     …
42.     <wormbase:Protein rdf:ID=“Protein 1”>
43.         <ann:OffsetOnHTMLPage/>10015< /ann:OffsetOnHTMLPage/>
44.         <ann:HTMLText/>WP:CE18608< /ann:HTMLText/>
45.         <wormbase:ProteinValue>WP:CE18608< /wormbase:ProteinValue>
46.         <wormbase:Protein-Genemodels rdf:resource=”#Genemodels 1”/>
47.     </wormbase:Protein>
48.     <wormbase:AminoAcids rdf:ID=“AminoAcids 1”>
49.         <ann:OffsetOnHTMLPage/>10152< /ann:OffsetOnHTMLPage/>
50.         <ann:HTMLText/>342 aa< /ann:HTMLText/>
51.         <wormbase:AminoAcidsValue>342 aa</wormbase:ProteinValue>
52.         <wormbase:AminoAcids-Genemodels rdf:resource=”#Genemodels 1”/>
53.     </wormbase:AminoAcids>
54.     …
55.     <wormbase:Protein rdf:ID=“Protein 2”>
56.         <ann:OffsetOnHTMLPage/>10689< /ann:OffsetOnHTMLPage/>
57.         <ann:HTMLText/>WP:CE28918</ann:HTMLText/>
58.         <wormbase:ProteinValue>WP:CE28918</wormbase:ProteinValue>
59.         <wormbase:Protein-Genemodels rdf:resource=”#Genemodels 2”/>
60.     </wormbase:Protein>
61.     <wormbase:AminoAcids rdf:ID=“AminoAcids 2”>
62.         <ann:OffsetOnHTMLPage/>10826< /ann:OffsetOnHTMLPage/>
63.         <ann:HTMLText/>406 aa< /ann:HTMLText/>
64.         <wormbase:AminoAcidsValue>406 aa</wormbase:ProteinValue>
65.         <wormbase:AminoAcids-Genemodels rdf:resource=”#Genemodels 2”/>
66.     </wormbase:AminoAcids>
67. …
68. </rdf:RDF>
```

Fig. 14. Annotation for Fig. 1 for the ontology in Fig. 12.

Line 45 tells its *ProteinValue*, also “WP:CE18608”. To identify the annotated string in the original page, TISP++ keeps track of the position where the values are located by recording the offset of the string from the beginning of the cached page, a local copy of the original page containing the values. Line 43 in Fig. 14, for example, shows that we can locate the value “WP:CE18608” at character 10015 in the cached HTML document.



```
PREFIX wormbase:<http://dithers.cs.byu.edu/owl/ontologies/wormbase#>
SELECT ?Protein ?AminoAcids
WHERE{
   ?Identification wormbase:Identification-IDs ?IDs ;
         wormbase:Identification-Genemodels ?Genemodels .
   ?IDs wormbase:IDs-CGCname ?CGCname .
   ?CGCname wormbase:CGCnameValue ?GeneName .
   ?Genemodels wormbase:Genemodels-Protein ?ModelsProtein ;
         wormbase:Genemodels-AminoAcids ?ModelsAminoAcids .
   ?ModelsProtein wormbase:ProteinValue ?Protein .
   ?ModelsAminoAcids wormbase:AminoAcidsValue ?AminoAcids .
    FILTER regex (?GeneName, "cdk-4")
 }
```

	Protein	AminoAcids
☐	WP:CE28918	406 aa
✓	WP:CE18608	342 aa

ipt)	Protein	Amino Acids
2	WP:CE18608	342 aa
2	WP:CE28918	406 aa

domain PF00069 (Protein kinase
90 (Serine/threonine protein
Tyrosine protein kinase),

Fig. 15. A sample SPARQL query and the results for the annotation in Fig. 14.

With the annotated data properly stored in an RDF file (e.g. Fig. 14), we are now ready to query the annotated data using SPARQL [33]. A simple query illustrates how this works. If we want to find the protein and the amino-acids information for gene "cdk-4", we can write the SPARQL query in Fig. 15. Fig. 15 shows the query and the result of this query.

The *FILTER* statement in the SPARQL query allows the variable *?GeneName* to bind only to values containing string "cdk-4". The *CGCnameValue* in Line 38 of Fig. 14 satisfies this constraint. Through the property *wormbase:CGCnameValue* (Line 35), the SPARQL query associates *?GeneName* values with *?CGCname* instance values. Thus, in our example, the instance value *CGCname_1* (Line 35) binds to the variable *?CGCname*. Then, through the property *wormbase*:*IDs-CGCname*, the query finds the corresponding instances of *IDs* (in our example, *IDs_1* in Line 18). Next, through the *wormbase*:*Identification-IDs* property, the query finds the instances of *Identification* that associate with *IDs_1* (in our example, *Identification_1* in Line 11). Thereafter, through the *wormbase*:*Identification-Genemodels* property, the query finds the instances *Genemodels_1* and *Genemodels_2* (Lines 14 and 15). The query then locates the result instances, finding *Protein_1*, *Protein_2*, *AminoAcids_1*, and *AminoAcids_2* through the properties *wormbase*:*Genemodels-Protein* and *wormbase*:*Genemodels-AminoAcids* (Lines 26, 31, 27, and 32). Finally, the query returns the result values for these instances through the properties *wormbase*:*ProteinValue* and *wormbase*:*AminoAcidsValue* (Lines 45, 58, 51, and 64).

A user can select one or more returned records by checking the corresponding check boxes. TISP++ then shows the user the original source page with the values of interest highlighted. Fig. 15 shows that we have selected the check box of the second record in the results. Thus, in the displayed page the values *WP:CE18608* and *342 aa* are highlighted. In addition to allowing a user to select a record of values, TISP++ also makes all values individually clickable. When a user clicks on an individual value, TISP++ displays the source page with the corresponding value highlighted.

As a result of TIPS++ processing, which includes table understanding, ontology generation, and semantic annotation, we make hidden-web data present in HTML tables completely accessible by a standard query system. In addition, TISP++ also semantically annotates the data with respect to the generated OWL ontology, so that the data becomes machine-understandable and automatically manipulatable by computer agents.

9. Related work

9.1. Sibling page comparison

Other researchers have also tried to take advantage of sibling pages. RoadRunner [8] 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 [2] 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 [48] compares different records in a page instead of sibling pages and tries to find the extraction template for the record. This approach first tries to find individual data records by using a few heuristics. It then uses a tree edit distance algorithm to compare different data records and tries to find the extraction region. The approach in [26] compares sibling pages to filter out general headers and footers and other constant non-data areas of a page. It then makes various comparisons among main pages and linked pages to find record segmentations.

TISP fundamentally differs from these approaches. The first three [2,8,48] focus on finding data fields, and the technique in [26] focuses on record segmentation. They do not discover labels or try to associate data and labels. TISP focuses on table interpretation. It looks for a table pattern in addition to data fields. Furthermore, TISP 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.

9.2. Table interpretation

Automated table processing is typically done in two steps: (1) table recognition—find the data table, and (2) table interpretation—find and associate labels and values. Recent surveys [15,47] describe the vast amount of research that has been done in table processing and illustrate the challenges of automated table processing. Most of this work is about tables in imaged documents, but some is about HTML tables. Since we focus in this paper only on HTML tables, we limit the related work we discuss to only HTML table processing.

Several researchers have tried to differentiate data tables from tables for layout [4,7,19,42]. They have tried to use machine learning methods [7,42], visual level features [19,20], and linguistic features [4]. TISP provides a unique way to do this task when sibling pages are available. By considering the match percentage between tables, we were able to filter out all the layout tables in the car and geopolitical domain and only failed to filter out three pairs of tables out of more than 800 HTML tables from the molecular biology domain (these three false positives were also filtered out during the process of pattern generation). The approaches [4,7,19,42] were able to achieve *F*-measures of 86.5% [4], 95.5% [7], 90.0% [19], and 87.6% [42]. By way of a comparison, TISP was able to achieve an *F*-measure of 99.4%. TISP techniques, of course, only work when sibling pages and tables are available. Further, the experimentation was for different data sets, so these comparative results should not be construed as being definitive. The results only give an indication of about where the techniques might stand with respect to each other.

Several papers have discussed the HTML table interpretation problem. Some table interpretation systems work based on simple assumptions and heuristics (e.g. [4,16,17,22,27]). These simple assumptions (labels are either the first few rows or the first few columns) are easily broken in complex tables such as nested tables (e.g. Fig. 1) or tables with combination structures (e.g. Fig. 8). The approach in [31] presents a table interpretation system for automatic generation of *F*-logic frames for tables. It considers many linguistic features in a table such as alphabetic features, numeric features, number ranges, and data formats. It calculates differences among different regions of a table to detect the orientation of a table and to locate label cells and value cells. The average *F*-measure of this approach is around 50%. The technique in [38] learns lexical variants from training examples and uses a vector space model to deal with non-exact matches among labels. It also uses a few heuristics to find the association among labels and values. It achieves an *F*-measure of 91.4%. The approach in [20] uses visual boxes instead of HTML tags to interpret HTML tables. It achieves an *F*-measure of 52.1% (the precision value was 57% and the recall value was 48%). By way of comparison, TISP is able to achieve an *F*-measure of 94.5%. Of course, TISP only works when sibling tables are available. On the other hand, when applicable, TISP has the advantage over machine learning because it is unsupervised and document and web-site independent. TISP has no need for training data and works for all domains and web sites where sibling pages with sibling tables are available. Here again, these comparative results should not be construed as being definitive since the various research groups used different data sets for experimentation. Furthermore, the measurements here are for solving variations of the same problem, not identical problems.

9.3. Ontology generation

In recent years, many researchers have tried to facilitate ontology generation. Manual editing tools such as Protégé [29] and OntoWeb [34] have been developed to help users create and edit ontologies. It is not trivial, however, to learn ontology modeling languages and complex tools in order to manually create ontological description for information repositories.

Because of the difficulties involved in manual creation, researchers have developed semi-automatic ontology generation tools. Most efforts so far have been devoted to automatic generation of ontologies from text files. Tools such as OntoLT [3], Text2Onto [6], OntoLearn [28], and KASO [43] use machine learning methods to generate an ontology from arbitrary text files. These tools usually require a large training corpus and use various natural language processing algorithms to derive features to learn ontologies. The results, however, are not very satisfactory [30].

Tools such as SIH [37], TANGO [39], and the one developed by Pivk [30] use structured information (HTML tables) as a source for learning ontologies. Structured information makes it easier to interpret new items and relations. The approach in [30] tries to discover semantic labels for table regions and generate an ontology based on a table's structure. But how this process is done and what format the generated ontologies have is not discussed in the paper. SIH [37] and TANGO [39] are two ongoing projects we are currently working on. SIH tries to generate user-specified ontologies depending on user-generated forms. TANGO generates ontologies by analyzing related tables in a specific domain, generating an ontology according to each table, and then merging these ontologies to a general ontology for the domain. TISP++ can generate OWL ontologies fully automatically. It, however, only generates an ontology for a single set of sibling pages. It does not merge ontologies generated from different web sites, nor does it provide for user-specified ontologies. In addition, TISP++ generates ontologies in only one simple way, while TANGO aims at generating more sophisticated ontologies.

9.4. Semantic annotation

Existing semantic annotation systems can be classified into pattern-based systems and machine learning-based systems. Pattern-based systems such PANKOW [5] and Armadillo [12] find entities by discovering patterns. The pattern are either discovered manually or induced semi-automatically with a set of initial manually tagged seed patterns. Systems such as SemTag [9], AeroDAML [25], and KIM [32] use a set of pre-defined rules to locate the information of interest. OWL-AA [10,11] uses a domain-specified extraction ontology to locate semantic entities. Systems such as S-CREAM [21] and MnM [40] use machine learning algorithms and natural language processing methods to locate semantic entities. All of these approaches require some pre-defined information. Pattern-based approaches need a set of initial seed patterns. Rule-based approaches need a set of pre-defined rules. Extraction ontology-based approaches need domain ontologies. And machine learning-based approaches need a training corpus. TISP++, however, does not require a training corpus or pre-defined domain knowledge and relies only on the definition of a few typical table-pattern templates, which are all domain-independent.

10. Conclusion and future work

In this paper we introduced TISP, an approach 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%.

We also extended TISP to TISP++. TISP++ uses TISP results to semantically annotate web pages, turning their embedded facts into externally accessible facts. Given an interpreted table, TISP++ automatically generates an OWL ontology depending on the table's structure and then semantically annotates the data in the table with respect to this generated ontology. By doing so, all the data present in the sibling tables becomes accessible through a standard query interface.

Several directions remain to be pursued. For TISP and table interpretation, we would like to do the following. (1) 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 Fig. 1. As a future work on table interpretation, we would like to analyze cell content to find structured information within cells. (2) Some web pages use lists as tables; we would like to consider them too. (3) We would also like to interpret non-HTML tables such as tables in plain text. (4) We would also like to be able to deal with the case in Fig. 11a, where we need to join adjacent HTML tables to form a single table, and we would like to improve TISP so that it can interpret tables with factored labels, where part of a label has been removed and placed in a higher level heading, such as those in Fig. 11b.

For TISP++ and semantic ontology generation and semantic annotation, we would like to do the following. (1) Average web users need a more user-friendly query system, so that they can find data of interest without knowing SPARQL. We plan to provide our users with a natural language-based query interface (e.g. like [1]) and a form-based query interface (e.g. like [14]). (2) We plan to allow users to harvest information from multiple sites according to a personalized view. Our users would have the option to choose which data in a table they want to include and how this data should be organized. (3) We would like to generate more sophisticated ontologies that cover more complicated situations such as *n*-ary relationships, generalization/specialization, and aggregation.

Acknowledgements

This work is supported in part by the National Science Foundation under Grant #0414644. We would like to thank Stephen W. Liddle, Yihong Ding, and Andrew Zitzelberger for their help and efforts in implementing the query system for TISP++.

Appendix. TISP algorithms

Algorithm 1. processTISP(list of two or more sibling pages)

```
 1: randomly select two sibling pages for initial pattern recognition
 2: parse both to a DOM tree, yielding $pageDOM_1$ and $pageDOM_2$
 3: find all the HTML tables in $pageDOM_1$ and $pageDOM_2$
 4: obtain unnested tables for both, yielding $tables_1$ and $tables_2$
 5: run simpleTreeMappingAlgorithm to produce a match score for each table in $tables_1$ with each table in $tables_2$
 6: run stableMarriageAlgorithm to produce potentialSiblingTablePairs, the best 1–1 mapping between $tables_1$ and
    $tables_2$
 7: save in unpairedTables any tables in $tables_1$ and $tables_2$ not paired in potentialSiblingTablePairs
 8: for each potentialSiblingTablePair in potentialSiblingTablePairs do
 9:   matchPercentage = 100 (match score for the potentialSiblingTablePair)/(size in terms of DOM – tree nodes of the
      smaller table in the pair)
10:   if lower threshold < matchPercentage < higher threshold then
11:     put the potentialSiblingTablePair in a list of siblingTablePairs
12:   else if higher threshold $\leqslant$ matchPercentage then
13:     add both tables to layoutTables
14:   else if matchPercentage $\leqslant$ lower threshold then
15:     add both tables to unpairedTables
```

Algorithm 1 (*continued*)

```
16:     end if
17: end for
18: let tablePatterns be an empty list of table patterns
19: for each siblingTablePair in siblingTablePairs do
20:     tablePattern = determineTablePattern(siblingTablePair) -- see Algorithm 2
21:     if tablePattern ≠ null then
22:         add tablePattern to tablePatterns
23:     end if
24: end for
25: let interpretedTables be an empty list of interpreted tables
26: for each sibling page in the input list do
27:     parse page to a DOM tree, yielding pageDOM
28:     interpretTables(pageDOM, tablePatterns, interpretedTables, unpairedTables, layoutTables) -- see Algorithm 3, which
        adds tables in pageDOM interpreted by patterns in tablePatterns to interpretedTables and may also adjust existing
        patterns in tablePatterns and may pair tables in pageDOM with tables in unpairedTables producing either addi-
        tional interpretedTables or additional layoutTables
29: end for
30: use tablePatterns to generate a conceptualization (e.g., generate an OWL ontology) -- see Section 7
31: use interpretatedTables to annotate pages (e.g., for the conceptualization, extract data and data-item offsets and
    generate RDF triples) -- see Section 8
```

Algorithm 2. determineTablePattern(*siblingTablePair*)

```
1: run simpleTreeMappingAlgorithm on siblingTablePair to identify matched nodes and mis-matched nodes in each
   table
2: for all pre-defined table patterns
3:     for all labels in the pattern
4:         if the nodes in the label position in the siblingTablePair are matched nodes then
5:             increase matchCount by 1
6:         end if
7:     end for
8:     for all values in the pattern
9:         if the nodes in the value position in the siblingTablePair are mis-matched nodes then
10:            increase matchCount by 1
11:        end if
12:    end for
13:    patternMatchPercentage = 100 * matchCount/(total number of leaf nodes in the two tables in siblingTablePair
   divided by two)
14: end for
15: find the pattern that has the highest patternMatchPercentage
16: if the highest patternMatchPercentage > threshold then
17:        create an xpath expression as the pattern's path; the xpath expression must accommodate paths from root to
   table in both sibling tables
18:    record the label names found as labels for the pattern
19:    mark any label that does not appear in both sibling tables as optional
20:    return the pattern, the xpath expression, the label names, and the optional indicators as the tablePattern
21: else if findCombination(siblingTablePair) ≠ null then
22:    return the result of findCombination(siblingTablePair) -- see Algorithm 5
23: else
24:    return null
25: end if
```

Algorithm 3. interpretTables(*pageDOM*, *tablePatterns*, *interpretedTables*, *unpairedTables*, *layoutTables*)

```
1: for each table in pageDOM do
2:     get the xpath expression for table in pageDOM
3:     use the xpath expression to identify table in layoutTables or in tablePatterns
4:     if table identified in layoutTables then
5:         do not interpret -- not a data table
6:     else if table identified in tablePatterns then
7:         let foundPattern be the table pattern found
8:         tableMatch = matchTablePattern(table, foundPattern) -- see Algorithm 4
```

(*continued on next page*)

Algorithm 3 (*continued*)

```
 9:     if tableMatch $\neq$ null then
10:         save returned interpreted table in interpretedTables
11:       else
12:         put table in unmatchedPathTables
13:       end if
14:     else
15:       put table in unmatchedPathTables
16:     end if
17: end for
18: -- The code above usually processes all tables in a page; the code below processes anomalies.
19: for each table in unmatchedPathTables do
20:     if table matches a table in layoutTables then
21:       adjust the xpath expression for the matching table in layoutTables
22:       do not interpret -- not a data table
23:     else if table matches with a pattern in tablePatterns then
24:       adjust the xpath expression for table in tablePatterns
25:       save in interpretedTables the interpreted table returned from having executed matchTablePattern to check the
    match with a pattern
26:     else
27:       put table in unmatchedTables
28:     end if
29: end for
30: for each table in unmatchedTables do
31:     check whether table is a sibling table of any table in unpairedTables
32:     if sibling table found then
33:       let s_table be the sibling table found
34:       if the match percentage is such that s_table is a layout table then
35:         add both table and s_table to layoutTables
36:         remove s_table from unpairedTables
37:       else
38:         tablePattern = determineTablePattern(<table,s_table>) -- see Algorithm 2
39:         if tablePattern $\neq$ null then
40:           save table pattern found in tablePatterns
41:           use the table pattern found to interpret both table and s_table
42:           save returned results for both tables in interpretedTables
43:           remove s_table from unpairedTables
44:         else
45:           add table to unPairedTables
46:         end if
47:       end if
48:     else
49:       add table to unpairedTables
50:     end if
51: end for
```

Algorithm 4. matchTablePattern(*table*, *pattern*)

```
 1: find all the potential label cells in table according to pattern
 2: for each potential label cell do
 3:     if it matches with the corresponding sibling cell then
 4:       increase matchCount by 1
 5:     end if
 6: end for
 7: find all the potential value cells in table according to pattern
 8: for each potential value cell do
 9:     if it fails to match with the corresponding sibling cell then
10:       increase matchCount by 1
11:     end if
12: end for
13: calculate matchRate = matchCount/(total number of cells in table)
14: if matchRate > threshold then
15:     for each extra label found do
16:       add the extra label to pattern, mark it optional
17:     end for
```

Algorithm 4 (*continued*)

```
18:   for each missing label found do
19:      mark it optional
20:   end for
21:   -- With respect to pattern, the cells in table are now known either to be label cells or value cells; further, it is
      known which labels associate with which values; all that remains to be done is to record this information.
22:   record label-value pairs in table with respect to pattern, yielding interpretedTable
23:   return interpretedTable
24: else
25:   return null
26: end if
```

Algorithm 5. findCombination(*siblingTablePair*)

```
1: let $table_1$ and $table_2$ be the two tables in siblingTablePair
2: let $tableRegions_1$ and $tableRegions_2$ be empty lists of table-row groups
3: for each row in $table_1$ do
4:   if (number of matched nodes in row)/(number of nodes in row) > threshold then
5:     mark row as a row of labels
6:   else
7:     mark row as a row of values
8:   end if
9: end for
10: if the first row of the table is not a row of labels then
11:   group the table rows from the first row up through the row just before the first row of labels or the end of the
      table
12:   add this table-row group to $tableRegions_1$
13: end if
14: for each row of labels do
15:   if there is at least one row of values between this row and the next row of labels or the end of the table then
16:     group the table rows from the row of labels through the row just before the next row of labels (or the end of the
        table)
17:     add this table-row group to $tableRegions_1$
18:   end if
19: end for
20: repeat 3–19 with $table_2$ and $tableRegions_2$
21: if the number of table regions in $tableRegions_1$ and $tableRegions_2$ is identical and nonzero then
22:   create tableRegionPairs of corresponding table regions in $tableRegions_1$ and $tableRegions_2$
23:   let tablePatternCombinations be an empty list of table pattern combinations
24:   for each tableRegionPair in tableRegionPairs do
25:     tablePatternCombination = determineTablePattern(tableRegionPair) -- see Algorithm 2
26:     if tablePatternCombination $\neq$ null then
27:       add tablePatternCombination to tablePatternCombinations
28:     end if
29:     if tablePatternCombinations is not empty then
30:       syntactically form the list into a single combined pattern
31:       if the combined pattern is the same as the combined pattern for the last time through findCombination then
32:         return null -- to prevent infinite recursion
33:       else
34:         return the combined pattern
35:       end if
36:     end if
37:   end for
38: end if
39: repeat 3–38 column-wise
40: return null -- neither row-wise nor column-wise pattern found
```

References

[1] M.J. Al-Muhammed, Ontology Aware Software Service Agents: Meeting Ordinary User Needs on the Semantic Web, Ph.D. Thesis, Brigham Young University, 2007.

[2] A. Arasu, H. Garcia-Molina, Extracting structured data from web pages, in: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD'03), San Diego, CA, 2003, pp. 337–348.

[3] P. Buitelaar, D. Olejnik, M. Sintek, Ontolt: a Protégé plug-in for ontology extraction from text based on linguistic analysis, in: Proceedings of the First European Semantic Web Symposium (ESWS'04), Heraklion, Greece, May 2004, pp. 31–44.

[4] H. Chen, S. Tsai, J. Tsai, Mining tables from large scale HTML texts, in: Proceedings of the 18th International Conference on Computational Linguistics (COLING'00), Saarbrücken, Germany, July–August 2000, pp. 166–172.

[5] P. Cimiano, S. Handschuh, S. Staab, Towards the self-annotating web, in: Proceedings of the 13th International Conference on World Wide Web (WWW'04), New York, NY, May 2004, pp. 462–471.

[6] P. Cimiano, J. Völker, Text2Onto—a framework for ontology learning and data-driven change discovery, in: Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems (NLDB'05), Alicante, Spain, June 2005, pp. 227–238.

[7] W.W. Cohen, M. Hurst, L.S. Jensen, A flexible learning system for wrapping tables and lists in HTML documents, in: Proceedings of the 11th International World Wide Web Conference (WWW'02), Honolulu, Hawaii, May 2002, pp. 232–241.

[8] V. Crescenzi, G. Mecca, 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), Rome, Italy, September 2001, pp. 109–118.

[9] S. Dill, N. Eiron, D. Gibson, D. Gruhl, R. Guha, A. Jhingran, T. Kanungo, K.S. Mccurley, S. Rajagopalan, A. Tomkins, A case for automated large-scale semantic annotation, Journal of Web Semantics: Science, Services and Agents on the World Wide Web 1 (1) (2003) 115–132.

[10] Y. Ding, D.W. Embley, S.W. Liddle, Automatic creation and simplified querying of semantic web content: an approach based on information-extraction ontologies, in: Proceedings of the First Asian Semantic Web Conference (ASWC'06), Beijing, China, September 2006, pp. 400–414.

[11] Y. Ding, D.W. Embley, S.W. Liddle, Enriching OWL with instance recognition semantics for automated semantic annotation, in: Proceedings of the First International Workshop on Ontologies and Information Systems for the Semantic Web (ONISW'2007), Auckland, New Zealand, November 2007, pp. 160–169.

[12] A. Dingli, F. Ciravegna, Y. Wilks, Automatic semantic annotation using unsupervised information extraction and integration, in: Proceedings of the Third International Conference on Knowledge Capture (K-CAP'03), Workshop on Knowledge Markup and Semantic Annotation, Sanibel Island, FL, October 2003.

[13] The W3C Architecture Domain, 2005. <http://www.w3.org/dom/>.

[14] D.W. Embley, NFQL: the natural forms query language, ACM Transactions on Database Systems 14 (2) (1989) 168–211.

[15] D.W. Embley, M. Hurst, D. Lopresti, G. Nagy, Table processing paradigms: a research survey, International Journal of Document Analysis and Recognition 8 (2–3) (2006) 66–86.

[16] D.W. Embley, C. Tao, S.W. Liddle, Automatically extracting ontologically specified data from HTML tables with unknown structure, in: Proceedings of the 21st International Conference on Conceptual Modeling (ER'02), Tampere, Finland, October 2002, pp. 322–327.

[17] D.W. Embley, C. Tao, S.W. Liddle, Automating the extraction of data from HTML tables with unknown structure, Data and Knowledge Engineering 54 (1) (2005) 3–28.

[18] D. Gale, L.S. Shapley, College admissions and the stability of marriage, American Mathematics Monthly 69 (1) (1962) 9–14.

[19] W. Gatterbauer, 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), Boston, MA, July 2006, pp. 1313–1318.

[20] W. Gatterbauer, P. Bohunsky, M. Herzog, B. Krüpl, 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, pp. 71–80.

[21] S. Handschuh, S. Staab, F. Ciravegna, S-CREAM—Semi-automatic CREAtion of Metadata, in: Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW'02), Siguenza, Spain, October 2002, pp. 358–372.

[22] W. Holzinger, B. Krüpl, M. Herzoge, Using ontologies for extracting product features from web pages, in: Proceedings of the Fifth International Semantic Web Conference (ISWC'06), Athens, GA, November 2006, pp. 286–299.

[23] P.G. Ipeirotis, L. Gravano, 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), Santa Barbara, CA, May 2001, pp. 67–78.

[24] Jena—A Semantic Web Framework for Java, 2008. <http://jena.sourceforge.net/>.

[25] P. Kogut, W. Holmes, AeroDAML: Applying information extraction to generate DAML annotations from web pages, in: Proceedings of the First International Conference on Knowledge Capture (K-CAP'01) Workshop on Knowledge Markup and Semantic Annotation, Victoria, British Columbia, 2001.

[26] K. Lerman, L. Getoor, S. Minton, C. Knoblock, Using the structure of web sites for automatic segmentation of tables, in: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data (SIGMOD'04), Paris, France, June 2004, pp. 119–130.

[27] S. Lim, Y. Ng, An automated approach for retrieving hierarchical data from HTML tables, in: Proceedings of the Eighth International Conference on Information and Knowledge Management (CIKM'99), Kansas City, MO, November 1999, pp. 466–474.

[28] R. Navigli, P. Velardi, A. Cucchiarelli, F. Neri, Quantitative and qualitative evaluation of the OntoLearn ontology learning system, in: Proceedings of the 20th International Conference on Computational Linguistics, Geneva, Switzerland, August 2004, pp. 1043–1050.

[29] N.F. Noy, M. Sintek, S. Decker, M. Crubezy, R.W. Fergerson, M. Musen, Creating semantic web contents with Protégé-2000, IEEE Intelligent Systems 16 (2) (2001) 60–71.

[30] A. Pivk, Automatic ontology generation from web tabular structures, AI Communications 19 (1) (2006) 83–85.

[31] A. Pivk, P. Cimiano, Y. Sure, From tables to frames, in: Proceedings of the Third International Semantic Web Conference (ISWC'04), Hiroshima, Japan, November 2004, pp. 166–181.

[32] B. Popov, A. Kiryakov, D. Ognyanoff, D. Manov, A. Kirilov, KIM—a semantic platform for information extraction and retrieval, Natural Language Engineering 10 (3–4) (2004) 375–392.

[33] SPARQL Query Language for RDF, 2008. <http://www.w3.org/TR/rdf-sparql-query/>.

[34] P. Spyns, D. Oberle, R. Volz, J. Zheng, M. Jarrar, Y. Sure, R. Studer, R. Meersman, OntoWeb—a semantic web community portal, in: Proceedings of the Fifth International Conference on Practical Aspects of Knowledge Management (PAKM'02), Vienna, Austria, December 2002, pp. 189–200.

[35] K.-C. Tai, The tree-to-tree correction problem, Journal of the ACM 26 (3) (1979) 422–433.

[36] C. Tao, D.W. Embley, Automatic hidden-web table interpretation by sibling page comparison, in: Proceedings of the 26th International Conference on Conceptual Modeling (ER'07), Auckland, New Zealand, November 2007, pp. 560–581.

[37] C. Tao, D.W. Embley, Seed-based generation of personalized bio-ontologies for information extraction, in: Proceedings of the First International Workshop on Conceptual Modelling for Life Sciences Applications (CMLSA'07), Auckland, New Zealand, November 2007, pp. 74–84.

[38] A. Tengli, Y. Yang, N.L. Ma, Learning table extraction from examples, in: Proceedings the 20th International Conference on Computational Linguistics (COLING'04), Geneva, Switzerland, August 2004, pp. 987–993.

[39] Y.A. Tijerino, D.W. Embley, D.W. Lonsdale, Y. Ding, G. Nagy, Toward ontology generation from tables, World Wide Web: Internet and Web Information Systems 8 (3) (2004) 251–285.

[40] M. Vargas-Vera, E. Motta, J. Domingue, M. Lanzoni, A. Stutt, F. Ciravegna, MnM: Ontology driven semi-automatic and automatic support for semantic markup, in: Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW'02), Siguenza, Spain, October 2002, pp. 379–391.

[41] X. Wang, Tabular Abstraction, Editing, and Formatting, Ph.D. Thesis, University of Waterloo, 1996.
[42] Y. Wang, 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), Honolulu, Hawaii, May 2002, pp. 242–250.
[43] Y. Wang, J. Völker, P. Haase, Towards semi-automatic ontology building supported by large-scale knowledge acquisition, in: AAAI Fall Symposium on Semantic Web for Collaborative Knowledge Acquisition, vol. FS-06-06, Arlington, VA, October 2006, pp. 70–77.
[44] Worm base! 2005. <http://www.wormbase.org>.
[45] XML Path Language (XPath), 2006. <http://www.w3.org/TR/xpath>.
[46] W. Yang, Identifying syntactic differences between two programs, Software Practice and Experience 21 (7) (1991) 739–755.
[47] R. Zanibbi, D. Blostein, J.R. Cordy, A survey of table recognition, International Journal of Document Analysis and Recognition 7 (1) (2004) 1–16.
[48] Y. Zhai, B. Liu, Web data extraction based on partial tree alignment, in: Proceedings of the 14th International Conference on World Wide Web (WWW'05), Chiba, Japan, May 2005, pp. 76–85.

Dr. Cui Tao received a B.S. degree in 1997 from Beijing Normal University, where she majored in Biology and minored in Computer Science. She recently received her Ph.D. in Computer Science from Brigham Young University. In 2009, Dr. Tao joined Mayo Clinic College of Medicine, Division of Biomedical Statistics and Informatics. Her research focuses on ontology generation, conceptual modeling, and information extraction over the biomedical domain. She is also interested in the Semantic Web and its application on biomedical and clinical data. Her research is supported partially by NIH and NCI. She is serving and has served as a PC member for various international conferences and workshops.

Dr. David W. Embley received a B.A. in Mathematics (1970) and an M.S. in Computer Science (1972), both from the University of Utah. In 1976 he earned his Ph.D. in Computer Science from the University of Illinois. From 1976 to 1982 he was a faculty member in the Department of Computer Science at the University of Nebraska, where he was tenured in 1982. Since then he has been a faculty member in the Department of Computer Science at Brigham Young University. He teaches graduate and undergraduate classes in database systems and theory, discrete mathematics, and extraction and integration of web data. He is co-director of the Data Extraction research group and has been co-director of the Object-oriented Systems Modeling (OSM) research group. He has published widely and has made numerous presentations at national and international conferences. His research is supported in part by the National Science Foundation. He is the author of "Object Database Development: Concepts and Principles," Addison-Wesley, Reading, Massachusetts, 1998, and a coauthor of "Object-oriented Systems Analysis: A Model-driven Approach," Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1992. He is a member of the steering committee for the International Conferences on Conceptual Modeling (the ER Conferences), and has served as chair for the committee. He is serving or has served in various other capacities, including as an editorial board member, PC chair, PC member, and workshop coordinator.