D.W. Embley, S. Seth, G. Nagy, Transforming Web tables to a relational database, Procs. ICPR 2014, Stockholm, Sweden, 2014.

Transforming web tables to a relational database

David W. Embley
Brigham Young University, Provo, UT, USA
embley@cs.byu.edu

Sharad Seth
University of Nebraska, Lincoln, NE, USA
seth@cse,unl.edu

George Nagy
Rensselaer Polytechnic Institute
Troy, NY, USA
nagy@ecse.rpi.edu

Abstract— HTML tables represent a significant fraction of web data. The often complex headers of such tables are determined accurately using their indexing property. Isolated headers are factored to extract category hierarchies. Web tables are then transformed into a canonical form and imported into a relational database. The proposed processing allows for the formulation of arbitrary SQL queries over the collection of induced relational tables.

Keywords—table segmentation; Wang categories; header paths; relational table SQL queries

I. Introduction

Recent estimates of the number of tables on the web range from hundreds of millions to billions. That is Big Data! Combining and querying this data is a tantalizing goal. To bring it closer, we propose two algorithmic techniques for structured pattern recognition of HTML and spreadsheet tables from heterogeneous sources. The first algorithm recognizes multi-column row headers and multi-row column headers by analyzing row, column and cell relationships in the entire table. It is an order of magnitude more accurate than previous table segmentation methods based on statistical classification of appearance features that represent only cell formatting. The second algorithm determines the often-hierarchical category structure that maps the 2-D table into a multi-category data cube.

These operations allow importing ordinary tables into a relational database in a canonical format that is expressive and flexible enough for arbitrary queries. We use the MS Access database system to demonstrate that the algorithmically processed HTML tables can be directly queried with SQL. We determine the latent table structure and execute queries with the following processing pipeline:

1. isolate row and column headers by locating the *minimum indexing point* of the table;
2. extract the category information required by a data cube view by *factoring* header labels;
3. transform the table to a *canonical* form that is agnostic as to which categories will be subsequently considered relational tuples or attributes;
4. import the canonical tables into a *relational database*;
5. formulate SQL *queries* on one or more tables.

To avoid parsing possibly idiosyncratic HTML code, we convert web tables to CSV tables that preserve their grid structure but lose most cell formatting information and unmerge all spanning cells. Although rendered versions of either file type can be readily parsed by human readers, neither representation explicitly ties the data (value) cells to their row and column headers. Since we don't rely on formatting information, we import arbitrary HTML tables from heterogeneous sources into MS Excel in CSV format, analyze them with Python programs, and then upload and query them in MS Access. We report experiments on 200 web tables from ten large statistical web sites from six countries on which we already have reliable ground truth and commensurable results.

The next section is a review of the most relevant previous work. Section 3 describes the indexing algorithm for locating table headers. Section 4 presents category extraction and canonical table generation. Section 5 gives an example of querying a table in Access. In Section 6 we summarize our results and propose related topics for further research.

II. PREVIOUS WORK

Wang and Hu, among others, have demonstrated successful methods for locating and delimiting HTML tables in spite of the common use of <table> tags for non-table page layout [1]. Parsing HTML coding can be avoided with Excel's built-in functions for importing HTML tables. The Office Excel 2007 XML-based xlsx file format preserves both structure and formatting [2]. The CSV format that we use retains the fundamental grid structure but not most cell formatting.

Pattern recognition, machine learning and image processing are standard approaches to segmenting and interpreting scanned images of printed tables and HTML files of web tables [3]. Previous segmentation methods typically located the boundary between headers and data cells using heuristics based on cell content and appearance for distinguishing headers from data cells and the rest of the table (e.g. table title and footnotes) [4,5, 6]. Such methods achieved 80-90% accuracy, but the formatting peculiarities causing the remaining errors vary enough to hamper further progress in this direction [7].

The immense variability of table vocabulary also results in diminishing returns from natural language processing [8,9]. Attempts at segmentation using table grammars—syntactic pattern recognition—did not give acceptable results either [10,11]. However, segmentation based on indexing, even

though more primitive than our current method, resulted in 98.5% accuracy [12]. The indexing property is fundamental and deserves to be incorporated in any table processing system aiming for high accuracy.

X. Wang introduced categories and header paths as constituents of a formal table data type for the Xtable editing and formatting system [13,14]. Most, if not all, previous table interpretation systems exploit only the geometric grid structure of header and data cells rather than the logical relationships induced by Wang-category-header indexes for a table. Category headers are often simple, but in general they are hierarchical forming category trees. Many tables have just two categories (2-D tables), but Wang points out that a table, in general, is a data cube with *n* category headers. The typical technique to render an *n*-D table ($n > 2$) on a 2-D plane is to create a cross product of the paths to the leaves of the category trees of one or both of the row or column headers [13,14]

Importing and querying visual tables in a Data Base Management System (DBMS) was originally proposed for scanned paper tables [15], and much later for Web tables [16]. A DBMS provides query and retrieval functions that allow combining information from several tables [17,18]. Although printed and HTML tables are logically symmetric in row and column organization, relational tables are not because their rows are *records* (or *tuples*), and their columns are *fields* (or *entities*). This distinction opens the way for a wealth of useful operations based on predicate logic and governed by the laws of relational algebra and calculus [19].

Earlier research on table processing primarily targeted scanned paper tables and ASCII tables (e.g. from email). Although much progress was achieved on segmentation based on rulings and on row and column alignment of unruled and ASCII tables, the OCR systems of that time could not cope with the structure of line breaks and spaces in scanned tables. (Current OCR systems do much better on tables, but their methods are proprietary.) A survey of research up to 2005 can be found in [20] which is, however, largely obsolete in view of recent work by teams sponsored by Google [21], Yahoo [22], Citeseer [23, 24] and by other academic groups [25, 26, 27].

Three recent (2013) papers are very much in the spirit of our work in aiming to convert tabular data to relational form. Adelfio and Samet [28] discover the table schema by conditional random fields (CRFs), adapting the technique originally used by Pinto et al. [29]. They classify each row as belonging to one of seven different classes (header, data, title, metadata, etc.) based on row features derived from the layout, style, and value attributes of the constituent cells. However, unlike the method proposed here, they cannot guarantee the validity of the discovered schema or identify the category hierarchy of the headers.

In contrast, extraction of the category hierarchy is one of the main contributions of Chen and Cafarella [30]. Their system pipeline is very similar to ours but with a radically different implementation. Unlike our algorithmic approach to segmentation (but like Adelfio and Samet), they adapt the CRF technique [29] to label each row with one of four labels: title, header, data, and footnote, using similar row features. Note that the rows labeled as "data" also include the cells in the row header, hence to distinguish between the two, they must assume that the data region is purely numeric. Their hierarchy extractor builds *ParentChild* candidates of cells in the header region using formatting, syntactic, and layout features. The candidate list is pruned by an SVM classifier that enforces the resulting set of candidate pairs to be cycle-free. In the algorithmic approach described in Section IV, the resulting structure is guaranteed to be cycle-free by construction.

Lautert et al. [31] formalize the notion of Web tables ("tabular structures found in Web pages, composed of an ordered set of x rows and y columns"), propose a primary and secondary taxonomy for *relational knowledge* tables, and describe an artificial neural network classifier to categorize Web tables. They find that only 17.75% of the Web tables in their huge collection have a relational structure, i.e. trivially convertible to relational database. We consider web tables with a row and column header structure that are inherently more difficult to transform to a relational form.

III. HEADER EXTRACTION BY INDEXING

The location of the boundary between headers and data cells by indexing is schematically illustrated in Fig. 1. The key aspect of this toy example is that different letters (and colors) stand for different symbol strings. The repeated entries result from left-filling blank cells from unmerged spanning cells. Each data cell is indexed by its row-header path and by its column header path. For example, the $100.00 cell is indexed by row-header path <K,L> and column-header path <A,C,D,G>. A well-formed table must have unique row-header paths and unique column-header paths so that the headers index the table. The task of the segmentation algorithm is to find a set of rows and columns that index the largest possible part of the rest of the table

•	A	A	A	A	A	A	A
	B	B	C	C	C	C	C
•	D	D	D	D	E	E	E
	H	F	G	H	F	G	H
K	K						
K	L		$100.00				
M	M						
M	N						

Fig. 1. Indexing by unique header paths of a 3-D 2×3×4 data-cube table.

Segmentation is accomplished by finding the Minimum Indexing Point that partitions the two headers and the stub head from the data region. We improved the Minimum Indexing Point Search (MIPS) reported in [12]. The algorithm never revisits exactly the same rows and columns and is therefore guaranteed to terminate. The only operation on cell contents is string comparison for exact equality.

The MIPS algorithm starts by checking if the first row and the first column suffice. The column paths are <A>, which obviously does not uniquely distinguish the columns. Adding the next row, it looks at <A,B>, <A,B>, <A,C>, <A,C>, …, which are also not unique. Adding the third row is still not enough because there are two <A,B,D> paths (and also

Table 9. Numbers of outgoing short messages and multimedia messages from mobile phones in 200					
Year	Short messages, thousands 1)	Change, %	Short messages/ subscription	Multimedia messages, thousands	Change, %
2003	1 647 218	24,3	347	2 314	
2004	2 193 498	33,2	439	7 386	219,2

Fig. 2. CSV version of part of a web table that requires "prefixing".

Table 9. Numbers of outgoing short messages and multimedia messages from mobile phones in 2002-2					
Year	Short messages, thousands 1)	Short messages, thousands 1)	Short messages/ subscription	Multimedia messages, thousands	Multimedia messages,
ditto	ditto	Change, %	ditto	ditto	Change, %
2003	1 647 218	24,3	347	2 314	
2004	2 193 498	33,2	439	7 386	219,2

Fig. 3. Prefixing repeated cell labels with a unique predecessor adds a row to the table.

repeated <A,C,D> and <A,C,E> paths). However, with four rows all the paths are unique: <A,B,D,H>, <A,B,D,F>, <A,C,D,G>, <A,C,D,H>, <A,C,E,F>, <A,C,E,G>, <A,C,E,H>.

Checking now the row header, the row paths <K>, <K>, <M>, <M> have duplicates. With the second column added, <K,K>, <K,L>, <M,M>, <M,N> are unique. But the algorithm must now backtrack to delete the first column of the column header.

We have now found indexing row and column headers, but the first two rows of the column header are redundant. The algorithm can eliminate these by traversing the header from the bottom up. Here it would stop at the third row because the paths through the third and fourth row suffice: <D,F>, <D,G>, <D,H>, <E,F>, <E,G>, <E,H>. This is part of our implementation only because our current queries don't use cell labels from redundant rows.

The program also finds empty rows or rows containing repetitive units above the data cells, and rows containing footnotes or other notes below the data cells. Therefore the data cell area is also completely demarcated.

Previous methods assumed that the grid layout of tables implies that all the header cells necessary to index a particular data cell are located directly to the left or directly above that data cell. This assumption fails on about 5% of our tables, as for example in the table of Fig. 2. The trouble is the repeated "Change, %". It is remedied by *prefixing* each multiple entry in the second row by the nearest preceding unique entry. Prefixing adds a row to the header, as shown in Fig. 3. It is often required in row headers where the indentation or boldface in the original web table (lost in the transition to CSV) makes the unique header paths obvious to humans.

When tested on 200 web tables from heterogeneous international sites our Python program found two tables that could not be indexed because of repeated rows. It segmented 198 tables correctly (with a non-fatal error on one table). The runtime of the program on 200 tables was 4 seconds on a venerable laptop. Our program successfully segmented, for example, the table in Fig. 2 and the more complex tables shown in Sections IV and V.

IV. CATEGORY EXTRACTION BY FACTORING

In our collection of 200 tables, sampled from large statistical websites in the US and abroad [32], a majority of row and column headers consist of a single category with a flat structure—only the leaves of the category tree. However, approximately 10% of our collection of tables has a multi-category header, and approximately 30% of the tables have a header with a truly hierarchical structure—i.e., having category trees with non-leaf header labels. These numbers are in the same range as those mentioned by Chen and Cafarella [30] for their WEB corpus of 410,554 spreadsheet tables. Fig. 1 shows an example of a multi-category header. It has two column categories and one row category:

ColCat_1	ColCat_2	RowCat_1
D	F	K
E	G	L
	H	M
		N

Categories are extracted by header "factoring" [33]. For example, the initial algebraic expression used for factoring the column header of the table in Fig. 1 is obtained by tracing the header paths from left to right:

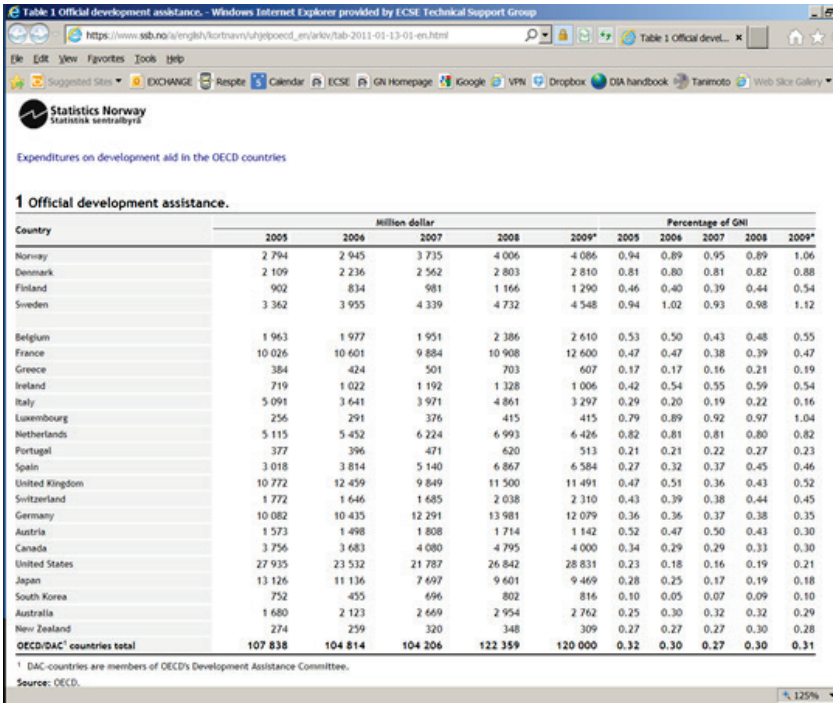
$$D*F + D*G + D*H + E*F + E*G + E*H$$

Note that the * and + operations in the expression represent vertical and horizontal concatenation, respectively and that the conventional operator precedence indicates the binding of the literals to the * and + operators. After factoring, the two non-singleton sum terms in the top-level product correspond to the column categories:

$$(D + E)*(F + G + H)$$

Similarly, the factoring of the row header yields the following single top-level sum term:

$$K*(K+L) + M*(M+N)$$



Statistics Norway
Statistisk sentralbyrå

Expenditures on development aid in the OECD countries

1 Official development assistance.

Country	Million dollar					Percentage of GNI				
	2005	2006	2007	2008	2009*	2005	2006	2007	2008	2009*
Norway	2 794	2 945	3 735	4 006	4 086	0.94	0.89	0.95	0.89	1.06
Denmark	2 109	2 236	2 562	2 803	2 810	0.81	0.80	0.81	0.82	0.88
Finland	902	834	981	1 166	1 290	0.46	0.40	0.39	0.44	0.54
Sweden	3 362	3 955	4 339	4 732	4 548	0.94	1.02	0.93	0.98	1.12
Belgium	1 963	1 977	1 951	2 386	2 610	0.53	0.50	0.43	0.48	0.55
France	10 026	10 601	9 884	10 908	12 600	0.47	0.47	0.38	0.39	0.47
Greece	384	424	501	703	607	0.17	0.17	0.16	0.21	0.19
Ireland	719	1 022	1 192	1 328	1 006	0.42	0.54	0.55	0.59	0.54
Italy	5 091	3 641	3 971	4 861	3 297	0.29	0.20	0.19	0.22	0.16
Luxembourg	256	291	376	415	415	0.79	0.89	0.92	0.97	1.04
Netherlands	5 115	5 452	6 224	6 993	6 426	0.82	0.81	0.81	0.80	0.82
Portugal	377	396	471	620	513	0.21	0.21	0.22	0.27	0.23
Spain	3 018	3 814	5 140	6 867	6 584	0.27	0.32	0.37	0.45	0.46
United Kingdom	10 772	12 459	9 849	11 500	11 491	0.47	0.51	0.36	0.43	0.52
Switzerland	1 772	1 646	1 685	2 038	2 310	0.43	0.39	0.38	0.44	0.45
Germany	10 082	10 435	12 291	13 981	12 079	0.36	0.36	0.37	0.38	0.35
Austria	1 573	1 498	1 808	1 714	1 142	0.52	0.47	0.50	0.43	0.30
Canada	3 756	3 683	4 080	4 795	4 000	0.34	0.29	0.29	0.33	0.30
United States	27 935	23 532	21 787	26 842	28 831	0.23	0.18	0.16	0.19	0.21
Japan	13 126	11 136	7 697	9 601	9 469	0.28	0.25	0.17	0.19	0.18
South Korea	752	455	696	802	816	0.10	0.05	0.07	0.09	0.10
Australia	1 680	2 123	2 669	2 954	2 762	0.25	0.30	0.32	0.32	0.29
New Zealand	274	259	320	348	309	0.27	0.27	0.27	0.30	0.28
OECD/DAC[1] countries total	**107 838**	**104 814**	**104 206**	**122 359**	**120 000**	**0.32**	**0.30**	**0.27**	**0.30**	**0.31**

[1] DAC-countries are members of OECD's Development Assistance Committee.
Source: OECD.

Fig. 4. Development Assistance Table from Norway Statistics, www.ssb.no/en/ posted in 2009.

RowCat_1.1	ColCat_1.1	ColCat_2.1	DATA
Norway	Million dollar	2005	2 794
Norway	Million dollar	2006	2 945
Norway	Million dollar	2007	3 735
Norway	Million dollar	2008	4 006
Norway	Million dollar	2009*	4 086
Norway	Percentage of GNI	2005	0.94
Norway	Percentage of GNI	2006	0.89
Norway	Percentage of GNI	2007	0.95
Norway	Percentage of GNI	2008	0.89
Norway	Percentage of GNI	2009*	1.06
Denmark	Million dollar	2005	2 109
Denmark	Million dollar	2006	2 236
Denmark	Million dollar	2007	2 562
Denmark	Million dollar	2008	2 803
Denmark	Million dollar	2009*	2 810
Denmark	Percentage of GNI	2005	0.81
Denmark	Percentage of GNI	2006	0.8
Denmark	Percentage of GNI	2007	0.81
Denmark	Percentage of GNI	2008	0.82
Denmark	Percentage of GNI	2009*	0.88
Finland	Million dollar	2005	902
Finland	Million dollar	2006	834
Finland	Million dollar	2007	981
Finland	Million dollar	2008	1 166

Fig. 5. Indexing by unique header paths (partial).

representing a single row category with a more complex structure. Factoring also sheds light on within-category header hierarchy, e.g. the K appearing in the left column is the parent of the K and L appearing to its right. Here, the category hierarchy is defined by its four attribute-value pairs (K,K), (K,L), (M,M), and (M,N), i.e. a single category tree with two sub-trees rooted at K and M.

A recursive algorithm carries out the factoring of initial header-paths expression using only the distributive law, a*(b+c) = a*b + a*c. An analysis of the problem shows that, apart from the base cases corresponding to a single sum or product term, four other forms of decomposition of the argument expression need to be considered during any call to the routine: (1) a*F, (2) a*F+G, (3) (a+b+…)*F, and (4) (a+b+…)*F + G, where a and b correspond header cell labels and F and G correspond to arbitrary algebraic expressions [33].

The table designer's choice of rows or columns for laying out the categories depends primarily on the number of items in the category and on the size and aspect ratio of the available space. In relational tables, however, rows are tuples (*records* in Access), while columns are attributes (*fields* in Access). The database schema immutably assigns the values of each variable to either a record or a field. We introduce canonical tables to bridge commonly accepted interpretations of "ordinary" tables and relational tables. Our canonical table is an M×1 relational table where each row comprises the indexing header paths and the corresponding indexed data value. Therefore the number of rows in the canonical table equals the number of data cells in the original table (plus one for the relational table's field names in a header row). Fig. 5 shows rows for the first 24 data cells of the M×1 relational table for the "ordinary" table in Fig. 4.

To form the M×1 relational table and import it into a relational database, each cell label in the original header paths becomes a key field value, and the data becomes a non-key field value. Figures 4 and 5 show an example. The row headers in Fig. 4 are values in the RowCat_1.1 column in Fig. 5 and the column headers are distributed as values in the ColCat_1.1 and ColCat_2.1 columns. When the combined row and column headers that uniquely index each data value in the DATA column also index the data values in the original table, as they do in Figs. 4 and 5, our algorithms have correctly recognized the table's pattern and thus have parsed and interpreted the table correctly.

V. RELATIONAL QUERY CONSTRUCTION FROM CANONICAL TABLES

We demonstrate the usefulness of our automated table interpretation algorithms by showing how to pose a meaningful SQL query over the derived M×1 relational table. Of course, the query is just one example of the multitude of queries that could be posed over either a single table or combinations of derived tables. The point is that by interpreting human-readable tables as relational tables, they become machine readable—queryable with SQL.

For the example query suppose that we wish to know for each year how far and in which direction Canada's percentage of GNI differs from the overall average for all countries. We import the M×1 table of Fig. 5 directly into

Access—without any editing. By default, Access names the table "Table1" and, although not necessary, we renamed it "DevAssistanceTable" for ease of readability.

Access automatically adjusts the names for fields by removing the dots since they violate the syntax requirements for Access field names. Access also automatically assigns types for the fields—text for the first three columns and double for the DATA column. (Note that the year 2009 has an asterisk, which disallows the field from having a numeric type, and that although all values in the DATA column are numeric, the space in the "million dollar" values prevents Access from properly interpreting them. It does, however, properly interpret the decimal "Percentage of GNI" numbers, rendering them as double-precision values.)

We can now directly pose the query in Figure 6, which yields the result in Figure 7. The query joins the M×1 table (named "DevAssistanceTable" upon import) with itself, as specified in the FROM clause—i.e., joins two instantiations of the table, t1 and t2, just as SQL can join any two relational tables. Next, it limits the cross product produced by the join to rows with "Canada" and "Percentage of GNI" values in the first instantiation of the table (t1), to the total rows with only percent-GNI values in the second instantiation of the table (t2), and to rows where the ColCat_21 values (the year values) are the same. The query then restricts the values in the remaining rows to those specified in the SELECT clause, all renamed with SQL's "as" syntax to be more readable. The query computes the DiffFromAve column by taking the difference in the DATA columns from the two instantiated tables, t1 and t2, for the rows that remain in the limited cross product. Thus, for each year, the query computes the difference between Canada's percentage of GNI and the average percentage of GNIs for all countries.

Query1 | DevAssistanceTable

```
SELECT t1.RowCat_11 as Country, t1.ColCat_21 as Year, t1.DATA as PercentGNI
  t2.DATA as OverallAvePercentGNI, t1.DATA - t2.DATA as DiffFromAve
FROM DevAssistanceTable t1, DevAssistanceTable t2
WHERE t1.RowCat_11 = "Canada"
  and t1.ColCat_11 = "Percentage of GNI"
  and t2.RowCat_11 = "OECD/DAC1 countries total"
  and t2.ColCat_11 = "Percentage of GNI"
  and t1.ColCat_21 = t2.ColCat_21;
```

Fig. 6. SQL Query.

Query1 | DevAssistanceTable

Country	Year	PercentGNI	OverallAvePercentGNI	DiffFromAve
Canada	2005	0.34	0.32	0.02
Canada	2006	0.29	0.3	-0.01
Canada	2007	0.29	0.27	0.02
Canada	2008	0.33	0.3	0.03
Canada	2009*	0.3	0.31	-0.01

Fig. 7. Query Results.

Two other examples of tables with complex formats that we have imported into ACCESS are shown in Figs. 8 and 9.

VI. Conclusion

The fundamental indexing property of a table is that every value cell is uniquely designated by its row-header path and its column-header path. Each header path is a sequence of cells contained in either the row or the column

Fig. 8. An almost two-category row header that requires prefixing "Total persons in households" and "Average number of persons in households".

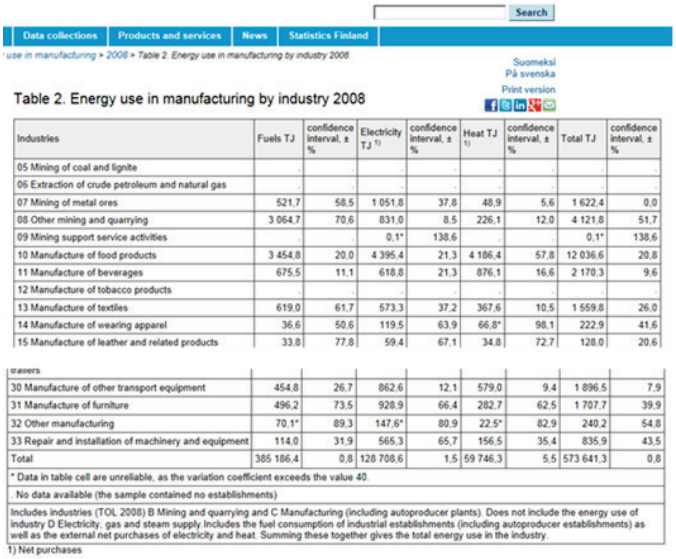


Fig. 9. The single-row column header has two categories of four (different) confidence intervals and four types of energy (including Total T.J.). In some cells dots serve as footnote markers.

header region of the table. Prefixing allows discovery of indirect header paths. Tables can be accurately segmented by an algorithm that exploits the indexing property. The table categories necessary for flexible choice of rows and columns in any relational database can be extracted from the segmented headers by factoring. The M×1 canonical table format circumvents the tuple/attribute dilemma. All of the above processing up to query generation is language and script independent. (Right-to-left scripts would only require changing the search direction for row headers.) The generation of canonical tables is fast enough for one thousand tables per hour even on a laptop.

We directly imported an M×1 table into the Access database system and posed a meaningful SQL query as an example of the usefulness of converting human-readable tables to machine- understandable tables. Once human-

readable tables are converted into relational tables, users can construct SQL queries over the database of generated relational tables—applying standard selection, projection, join, pivot, arithmetic, and aggregate operations.

Adaptation of these methods to scanned tables is subject only to OCR accuracy. PDF often scrambles page layout, therefore PDF tables may have to be rendered before reprocessing for importation into a DBMS.

We have already developed a format for auxiliary data like table title, footnote markers, footnote reference markers and footnote text. Table titles are usually found in the first or second cell of the first row. Footnotes are more complex. We first search for footnote markers below the table proper, separate them from the footnote text, and then search for the corresponding footnote reference markers in the header and data regions.

In the future we plan to clean up the automatic transformation of interpreted tables to relational database tables. For a fully automatic system, better and more flexible constant recognizers for number and date-time types are necessary. Further, instead of generic names such as "Table1". and "ColCat_1.1", meaningful labels like "DevAssistanceTable", "Year", and "Country" must be determined and applied.

We also plan to identify aggregates that so often appear in tables as computed data values In the past, the potential combinatorial explosion in the search for the aggregates has been dealt with by using linguistic clues (header words like "Total"). Since it is not uncommon to find aggregates that are not tagged with such labels, more sophisticated approaches are needed. The category structure may provide an elegant way to limit the search for potential aggregates.

ACKNOWLEDGMENT

Professor M.K. Krishnamoorthy (RPI) made significant contributions to our analysis of tables and to checking of results.

REFERENCES

[1] Y. Wang, and J. Hu, "Detecting tables in HTML documents," Procs. Int'l Wks. Document Analysis Systems, pp. 249–260, 2002.

[2] XLSX file format: http://office.microsoft.com/en-us/excel-help/file-formats-that-are-supported-in-excel-HP010014103.aspx#BMexcel

[3] S. Ferilli, Automatic Digital Document Processing and Management, Springer-Verlag, London 2011.

[4] C. Peterman, C.H. Chang, H. Alam, "A system for table understanding," Procs. Symposium on Document Image Understanding Technology, pp. 55–62, 1997.

[5] G. Nagy, S. Seth, D.W. Embley, M. Krishnamoorthy, D. Jin, S. Machado, "Data extraction from web tables: the devil is in the details," Procs. Int'l Conf. Document Analysis and Recognition, 2011.

[6] N. Di Mauro, F. Esposito, and S. Ferilli, "Finding critical cells in web tables with SRL: trying to uncover the devil's tease," Procs. Int'l Conf. Document Analysis and Recognition, 2013.

[7] G. Nagy, "Learning the characteristics of critical cells form web tables," Procs. Int'l Conf. Pattern Recognition, 2012.

[8] S. Douglas, M. Hurst, D. Quinn, "Using natural language processing for identifying and interpreting tables in plain text," Procs. Symposium on Document Analysis and Information Retrieval, pp. 535–545, 1995.

[9] M. Hurst, "Layout and language: beyond simple text for information interaction—modelling the table," Procs. Second Int'l Conf. Multimodal Interfaces, 1999.

[10] E.A. Green, M. Krishnamoorthy, "Recognition of tables using table grammars," Procs. Symposium on Document Analysis and Information Retrieval, pp. 261–277, 1995.

[11] S. Seth, R. Jandhyala, M. Krishnamoorthy, G. Nagy, "Analysis and taxonomy of column header categories for web tables," Procs. Int'l Wks. Document Analysis Systems, pp. 81–88, 2010.

[12] S. Seth, G. Nagy, "Segmenting tables via indexing of value cells by table headers," Procs. Int'l Conf. Document Analysis and Recognition, 2013.

[13] X. Wang, D. Wood, "Xtable: a tabular editor and formatter," Electronic Publishing, vol. 1, nr. 1, pp. 1–4, 1996.

[14] X. Wang, Tabular Abstraction, Editing, and Formatting, Doctoral Dissertation, University of Waterloo, 1996.

[15] T. Kieninger, A. Dengel, "A paper-to-HTML table converting system," Procs. Int'l Wks. Document Analysis Systems, 1998.

[16] W. Gatterbauer, P. Bohunsky, M. Herzog, B. Krupl, B. Pollak, "Towards domain-independent information extraction from web tables," Procs. 16th Int'l Conf. World Wide Web, pp. 71–80, 2007.

[17] D. Maier, The Theory of Relational Databases, Computer Science Press Inc., Rockville, MD, 1983.

[18] J. J. Adamski and K.T. Finnegan, Microsoft Access 2010, Course Technology, Boston, MA, 2011.

[19] J.D. Ullman, Principles of Database Systems, Computer Science Press Inc., 1980.

[20] D.W. Embley, M. Hurst, M. Lopresti, G. Nagy, "Table processing paradigms: a research survey," J. Doc. Anal. Recog. Vol. 8, nrs. 2–3, pp. 66-86, 2006.

[21] P. Venetis et al., "Recovering semantics of tables on the web," Procs. VLDB Endowment, vol. 4, nr. 9, pp. 528–538, 2011.

[22] N. Dalvi, R. Kumar, B. Pang, R. Ramakrishnan, A. Tomkins, P. Bohannon, S. Keerthi, S. Merugu, "A web of concepts, PODS, 2009.

[23] Y. Liu, K. Bai, P. Mitra, C.L. Giles, "TableSeer: automatic table metadata extraction and searching in digital libraries," Procs. 7th ACM/IEEE-CS Joint Conf. Digital Libraries, pp. 91–100, 2007.

[24] J. Fang, P. Mitra, Z. Tang, and C.L. Giles, "Table header detection and classification," Procs. 26th AAAI Conf. Artificial Intelligence, pp. 599–605, 2012.

[25] E.C. Silva, A.M, Jorge, L. Torgo, "Design of an end-to-end method to extract information from tables," Int. J. Doc. Anal. Recog. vol. 8, nr. 2, Springer, pp. 144–171, 2006.

[26] V. Long. An Agent-based Approach to Table Recognition and Interpretation, Mcquarie University, PhD disseration, 2010.

[27] G. Limaye, S. Sarawagi, S. Chakrabarti, "Annotating and searching web tables, using entities, types, and relationships," Procs. VLDB Endowment, vol. 3, nrs. 1–2, pp. 1338–1347, 2010.

[28] M. D. Adelfio and H. Samet, "Schema extraction for tabular data on the web," Procs. VLDB Endowment, vol. 6, nr. 6, pp 421–432, 2013.

[29] D. Pinto, A. McCallum, X. Wei, and W. B. Croft, "Table extraction using conditional random fields," Procs. SIGIR, pp. 235–242, 2003.

[30] Z. Chen and M. Cafarella, "Automatic web spreadsheet extraction," Procs. 3rd Wks. Semantic Search Over the Web, pp. 1–8, 2013.

[31] L. Lautert, M.M. Scheidt, C.F. Dorneles, "Web table taxonomy and formalization," SIGMOD Record, vol. 42, nr. 3, pp. 28–33, 2013.

[32] R. Padmanabhan, R.C. Jandhyala, M. Krishnamoorthy, G. Nagy, S. Seth, and W. Silversmith, "Interactive conversion of large web tables," Int'l Wks. Graphics Recognition, pp. 25–36, 2009.

[33] D.W. Embley, M. Krishnamoorthy, G. Nagy, S. Seth, "Factoring web tables," Procs. 24th EIA/AIE Conf., 2011.