Table Structure Recognition and Its Evaluation

Jianying Hu[a] and Ramanujan Kashi[a] and Daniel Lopresti[b] and Gordon Wilfong[b]

[a]Avaya Labs, 600 Mountain Ave, Murray Hill, NJ 07974

[b]Bell Labs, Lucent Technologies, 600 Mountain Ave, Murray Hill, NJ 07974

ABSTRACT

Tables are an important means for communicating information in written media, and understanding such tables is a challenging problem in document layout analysis. In this paper we describe a general solution to the problem of recognizing the structure of a detected table region. First hierarchical clustering is used to identify columns and then spatial and lexical criteria to classify headers. We also address the problem of evaluating table structure recognition. Our model is based on a directed acyclic attribute graph, or table DAG. We describe a new paradigm, "random graph probing," for comparing the results returned by the recognition system and the representation created during ground-truthing. Probing is in fact a general concept that could be applied to other document recognition tasks and perhaps even other computer vision problems as well.

Keywords: table recognition, document layout analysis, hierarchical clustering, document understanding, graph matching, computer vision, pattern recognition, performance evaluation

1. INTRODUCTION

Tables are an important means for communicating information in written media, and understanding such tables is a challenging problem in document layout analysis. In an earlier paper, we addressed the problem of detecting tables in multiple media.[1] The current work builds on this earlier work by recognizing the structure of the detected table. This forms part of a prototype we have implemented for a complete, end-to-end table understanding system that takes raw ASCII text as input, detects and parses any number of tables that it might contain, and generates a simple interactive man-machine dialog allowing a user to access the table data via a spoken language interface.[2]

For table structure recognition, a number of papers report on methods for determining the layout structure that rely solely on separator features such as vertical and horizontal lines or column spacing to segment the table into a structure of cells.[3,4] Others also use OCR results to aid in the segmentation of the table into regions such as *body*, *title block*, *column headers* and *row labels*.[5] The work by Hurst and Douglas is concerned with taking a segmented table and using the contents of the resulting cells to determine the logical structure of the table.[6]

A key component of table structure recognition is column segmentation. Columns are the most visually dominant structural components of a table. Entries in the body of a table are usually laid out in such a way that they can be visually segmented into distinct groups, each of which spans roughly the whole table body along the vertical direction. Among previous algorithms for table column segmentation without ruling lines, the most common are the vertical projection profile based methods.[7,8] Roughly speaking, in these methods the percentages occupied by white space (or alternatively black foreground) along evenly spaced vertical lines are computed. Then the resulting histogram is analyzed to search for peaks (or valleys when foreground histogram is used), which roughly correspond to gaps between columns. The problem with this approach is that when the columns are not perfectly aligned, or when the gaps between columns are slanted, the resulting histogram suffers from spurious peaks or flattened or completely smeared peaks, causing the results to be unreliable. A method based on $LR(k)$ parsing has been described, however it only works for a given class of tables (financial tables).[9] Kieninger proposed a bottom-up approach where vertically overlapping words are grouped into blocks.[10] Various complex heuristics are then applied to split or merge the blocks into proposed columns. Because of its pure bottom-up nature and the fact that the heuristics are based on very local analysis as well, the method suffers when there is "disruption" to the "normal" structure of the columns (e.g., the presence of a comment line spanning multiple columns, or two cells from different columns accidentally overlapping with each other), or when there is a gap within a column (e.g., the table in Figure 2). Recently, Ng

Email:{`jianhu,ramanuja`}`@avaya.com` {`dpl,gtw`}`@research.bell-labs.com`

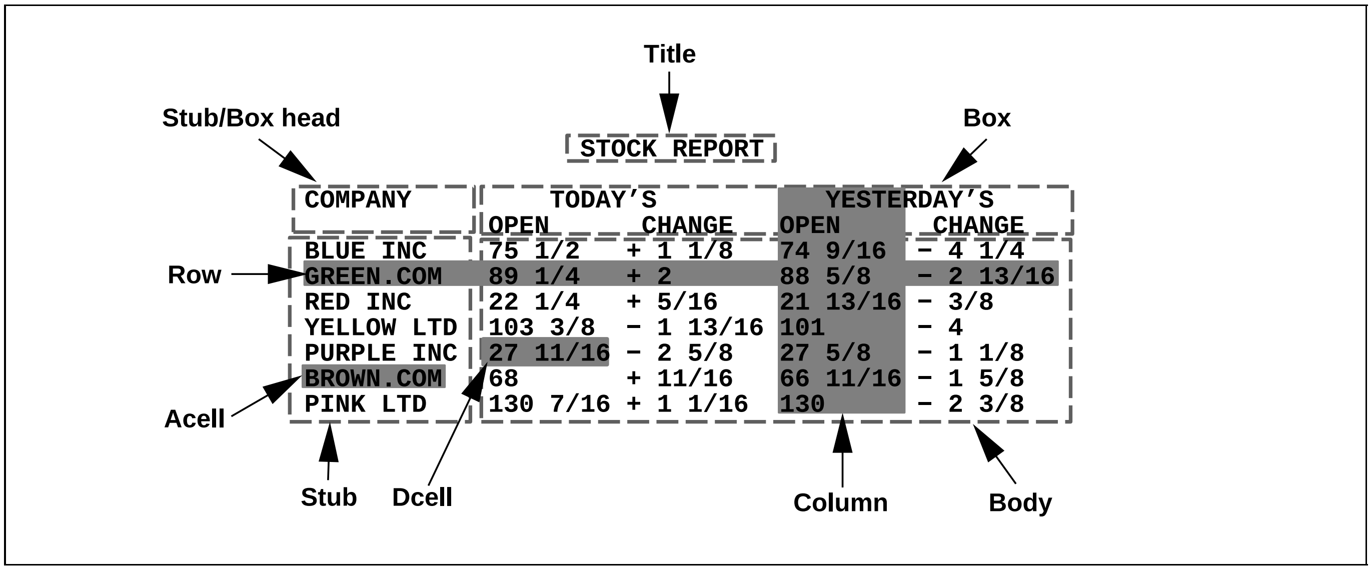
et al. proposed a machine learning based method for both table detection and column/row segmentation.[11] They designed a set of features for each subproblem and trained classifiers on a specific set of documents (Wall Street Journal news articles). The performance of these classifiers relies largely on the choice of features and the quality of training data. It is not clear whether the proposed features can generalize to documents in other domains.

In this paper, we describe a technique for recognizing the structure of a detected table region based on hierarchical clustering to identify columns and spatial and lexical criteria to classify headers. We also present a new paradigm we call "random graph probing" to evaluate the performance of table parsing algorithms. Preliminary experiments on a subset of the Wall Street Journal (WSJ) database and a small collection of email messages have produced promising results.

2. TABLE STRUCTURE RECOGNITION

The goal of recognition is to determine the structure of a given table and identify functional elements such as columns, rows, headers, etc. Many different terminologies have been used before by various researchers. We adopted one close to that proposed in Wang's Ph.D. thesis.[12] Defining a table model is itself a difficult issue, since both logical and layout conventions of tables vary depending on the document type, the subject domain and the medium.[13] Clearly, no model will be able to represent all possible tables. We chose to base our model on Wang's formalism because it provides a clean separation of content (logical model) from form (physical/presentational model), offers a rigorous mathematical representation as the logical model, and allows a large amount of flexibility.

Figure 1 illustrates the terminology used in this paper. At the lowest level, a table contains two types of cells: *Dcells* for data cells and *Acells* for access cells. These cells are organized into columns and rows. The column headers are grouped into a region named *box* and the row headers are grouped into a region called *stub*. The header for box/stub is called a *box head* or a *stub head*. The collection of all the Dcells comprises the *body*. The body is the only required region of a table. Acells and all header regions are optional.



STOCK REPORT

COMPANY	TODAY'S		YESTERDAY'S	
	OPEN	CHANGE	OPEN	CHANGE
BLUE INC	75 1/2	+ 1 1/8	74 9/16	– 4 1/4
GREEN.COM	89 1/4	+ 2	88 5/8	– 2 13/16
RED INC	22 1/4	+ 5/16	21 13/16	– 3/8
YELLOW LTD	103 3/8	– 1 13/16	101	– 4
PURPLE INC	27 11/16	– 2 5/8	27 5/8	– 1 1/8
BROWN.COM	68	+ 11/16	66 11/16	– 1 5/8
PINK LTD	130 7/16	+ 1 1/16	130	– 2 3/8

Figure 1. Table terminology (adapted from Wang's Ph.D. thesis).

2.1. Column Segmentation

The input to the column segmentation algorithm is the detected table region. It is assumed that this region contains all or nearly all the lines occupied by the body of the table. Depending on the layout of the particular table, it could also contain lines from column headers. Clearly if the the table has row headers, they are included in the region as well. At this point, no distinction is made between the column of row headers (stub) and a "normal" column.

Hierarchical clustering is applied to all words in this region to identify their likely groupings. Such groupings are represented as a binary tree, where the leaves represent words, the root represents the whole body, and the intermediate nodes represent nested groupings at different levels. This cluster tree is constructed in a bottom-up manner. First the leaf level clusters are generated, where each word belongs to a unique cluster. Then the two clusters with the minimum inter-cluster distance are merged into a new cluster. This new cluster is then represented by a new interior node with the two original clusters as its children. The merging process is repeated recursively until there is only one cluster left without a parent, which is then represented by the root node.

Efficient algorithms for general hierarchical clustering can be found in many text books.[14] There are two choices to be made in a particular application: how to define the distance between any two basic elements, and how to define the inter-cluster distance. In our application the basic elements are words. Since columns are grouped along the roughly vertical direction, we represent each word w_i for ASCII text by its starting and ending horizontal positions represented by the position vector $p_i = (s_i, e_i)$, also called *horizontal span* or simply *span*. The distance between two words w_i and w_j is then defined as the Euclidean distance between the two position vectors p_i and p_j. Note that the current definition represents the spatial distance between words in ASCII documents. Other types of distances, such as syntactic (e.g., a numeric string is different from an alphabetic string) or semantic (e.g., the numeric string representing a year is different from the numeric string representing a stock price) distances are also potentially useful. For inter-cluster distance computation, we chose to use the so called "average link." In other words, the distance between two clusters is computed as the average of the distances between all inter-cluster pairs of words.

The cluster tree generated in the above manner represents the hierarchical structure of the table body in terms of vertical grouping of words. Each cut across the tree provides one way of clustering these words. We need to find the cut such that each resulting cluster corresponds to a column. Such a cut is called the *column cut*.

The column cut is found using a breadth-first traversal of the cluster tree starting from the root. Two queues of nodes are maintained, one labeled *closed* and initialized to be empty, and the other labeled *open* and initialized to contain only the root node. Each node in the open queue is removed and examined to see if it should be further split. If the decision is "yes," then its children are pushed onto the end of the open queue. If the decision is "no," then the node is pushed onto the closed queue. The process continues until the open queue becomes empty, at which point the nodes in the closed queue represent the proposed columns.

Clearly the crucial step in generating the column cut is to decide if a node should be split. A poor decision would lead to either split or merged columns. To help make this decision, we define a measurement of the spacing, called *inter-cluster gap*, between two clusters C_1 and C_2 as described next. For each line containing words from both clusters, compute the minimum gap between a word from C_1 and a word from C_2. The median of all the minimum gaps is then defined as the the inter-cluster gap between C_1 and C_2. By definition, a leaf node cannot be split further. The decision as to whether a non-leaf node should be split is made using the following heuristics. Assuming the inter-cluster gap between the two children of a node is g:

1. The root node is always split, based on the assumption that a table has at least two columns.

2. The node is split if g is larger than or equal to a predefined constant G (currently set to 2). The assumption here is that a large enough gap always indicates column separation.

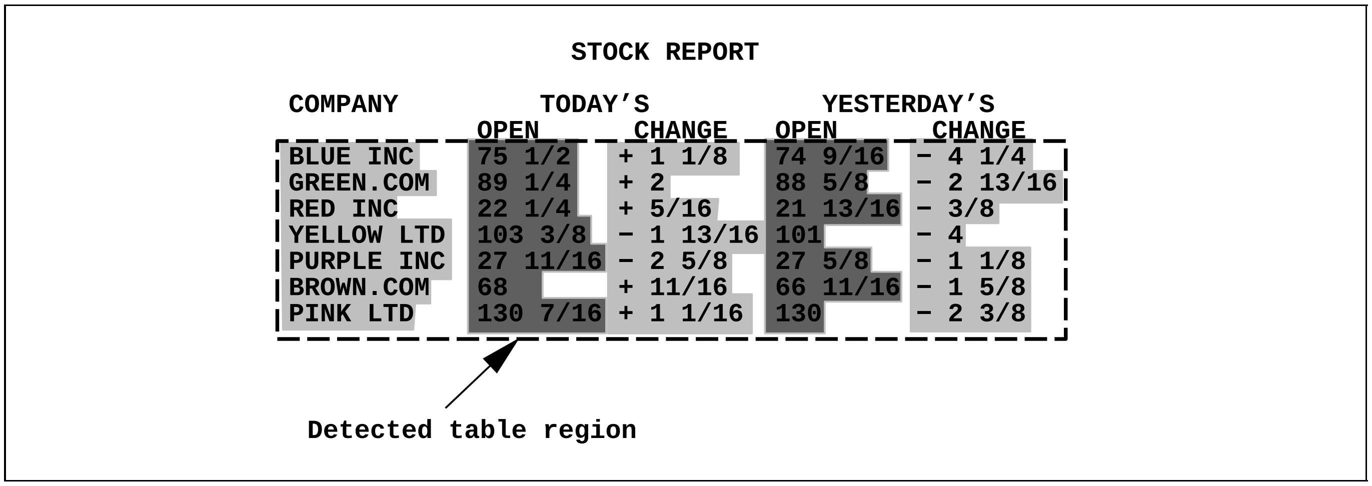
3. The node is also split if $g < G$, but $g/m_g > \alpha$, where m_g is the average inter-cluster gap between adjacent pairs of already identified columns (nodes in the closed queue), and α is a number between 0 and 1.0. The idea is that a small gap could also indicate column separation if such small gaps occur repeatedly.

4. The node is not split if it does not satisfy at least one of the above three conditions.

Note that even though the distance between words is currently defined only in terms of horizontal distance, the nature of the hierarchical clustering algorithm insures that the column segmentation algorithm proposed can handle imperfect vertical alignment very well. This ability is demonstrated in Figure 2 with a table containing ragged columns as well as columns with a small gap inside. As shown in the figure, all columns are clustered properly.

2.2. Header Detection and Row Segmentation

As mentioned earlier, headers including box (containing column headers), stub (containing row headers) and stub/box head are all considered optional. The current algorithm identifies potential headers based on spatial and some simple syntactic rules.


STOCK REPORT

COMPANY	TODAY'S		YESTERDAY'S	
	OPEN	CHANGE	OPEN	CHANGE
BLUE INC	75 1/2	+ 1 1/8	74 9/16	− 4 1/4
GREEN.COM	89 1/4	+ 2	88 5/8	− 2 13/16
RED INC	22 1/4	+ 5/16	21 13/16	− 3/8
YELLOW LTD	103 3/8	− 1 13/16	101	− 4
PURPLE INC	27 11/16	− 2 5/8	27 5/8	− 1 1/8
BROWN.COM	68	+ 11/16	66 11/16	− 1 5/8
PINK LTD	130 7/16	+ 1 1/16	130	− 2 3/8

Detected table region

Figure 2. A detected table region and its column segmentation using hierarchical clustering.

After column segmentation, the columns are first sorted according to their starting position, then the upper boundary of the detected table region is adjusted through a consistency check. The reason for this step is that since column headers do not always have perfect alignment with the columns, the table region delineated by the table detection algorithm may or may not include the headers. In order to get a consistent starting point, a correction step is applied to find the exact boundary between table body and potential headers. The adjustment is carried out using a lexical distance measure. We first define two types of strings: a string is considered *alphabetic* if it contains mostly alphabetical characters and *non-alphabetic* otherwise. A line is consistent with the columns if most of the words in the line are of the same type as the dominant type of its overlapping column, otherwise it is inconsistent. We define B to be the maximum possible number of lines in a box (currently set to 5). First the top B lines of the proposed table body are examined one by one. If the top-most line is inconsistent, than the maximal list of consecutive inconsistent lines from the top are removed from the body. Otherwise, the B lines directly above the table are examined one by one starting from the bottom. Each consistent line is added to the body until an inconsistent line is found.

After correction, we assume that the upper boundary of the table body correctly separates the headers from the body. Then, a region directly preceding the table body is identified as the potential box zone. The lower boundary (inclusive) of the zone is the lowest non-empty line above the first line of the table body and the upper boundary (inclusive) of the zone is set to be the line below the lowest empty line above the lower boundary, or B lines above the lower boundary, whichever is lower. This zone is then searched for potential column headers.

The following layout conventions for column headers hold for most tables we have encountered: (1) the header for each column is roughly aligned with the column; (2) hierarchical headers are placed such that the high level header is above its subsidiary headers and centered horizontally with regard to the columns represented by the subsidiary headers. Based on these observations, the lines in the potential box zone are examined one by one from the lower boundary to the upper boundary. First a line is segmented into phrases. This is currently carried out by simply considering a string of at least two consecutive spaces as phrase separators. Then the list of associated columns for each phrase is computed by searching for the maximal list of consecutive columns such that the span of each column in the list overlaps the span of the phrase. The following heuristics are then used to judge if a line is a header line: (1) every phrase in a header line must be associated with at least one column; (2) if a phrase in a header line is associated with more than one column, then each subsidiary column must already have its own header assigned. To capture the potential hierarchical structure, headers are represented by a tree structure which is initialized with the root representing the box, and k leaf nodes corresponding to the k columns. We define the *joint span* of a list of n spans $p_i = (s_i, e_i), i = 1 \ldots n$ as $p_{1 \cdot n} = (min(s_i, i = 1 \ldots n), max(e_i, i = 1 \ldots n))$. Once a higher level header is found, the corresponding intermediary node is generated, and the joint span of its subsidiary nodes is used to analyze the next line. Figure 3 shows the box of the table in Figure 1 represented as a tree. This tree is then traversed to assign headers to each columns (higher lever headers are shared by more than one column).

The following assumption is made regarding row headers: row headers, if present, are always contained in the left-most column of the table. In other words, unlike column headers which could be laid out hierarchically in

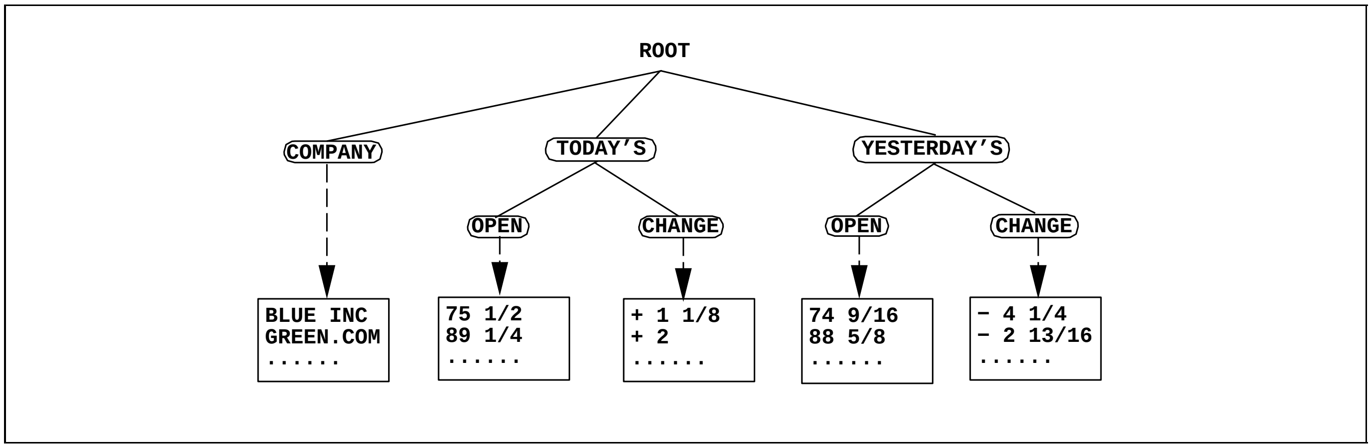
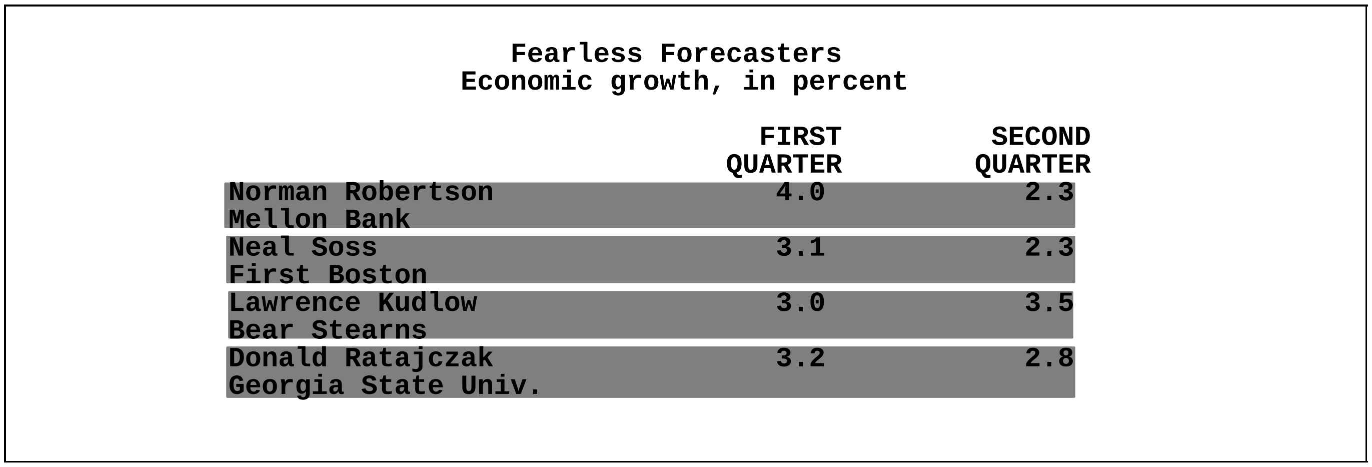


Figure 3. The tree representation of the box of the table in Figure 1.

separate horizontal layers, row headers are always arranged in a single vertical layer due to the physical constraints in arranging ASCII tables. If after column header detection the header of the left-most column is missing then the column is labeled stub. On the other hand, if the header of the left-most column is present the case is ambiguous; it could be a common column or it could also be stub with the stub head on top. We currently label them all as the latter since it appears that the left-most column of most tables can be interpreted as stub. A more precise interpretation can only be achieved with deep semantic analysis of the table elements.

Row segmentation is carried out after header detection. The difficulty here is that some or all of the cells in a table row could fall on more than one line and there is often no obvious separator between rows. The following heuristics have been adopted in our algorithm: (1) a blank line is always a row separator; (2) if a line contains non-empty strings for the stub (if it exists) and at least one other column, or if it contains non-empty strings for a majority of columns, then it is considered a *core line*, otherwise it is considered a *partial line*. Each table row contains one and only one core line and a partial line is always grouped with the core line above it. Occasionally there are tables where partial lines are grouped with the core line below. Such cases could be detected using statistical syntax analysis, such as N-grams. Figure 4 shows a table with multi-line rows and the row segmentation.

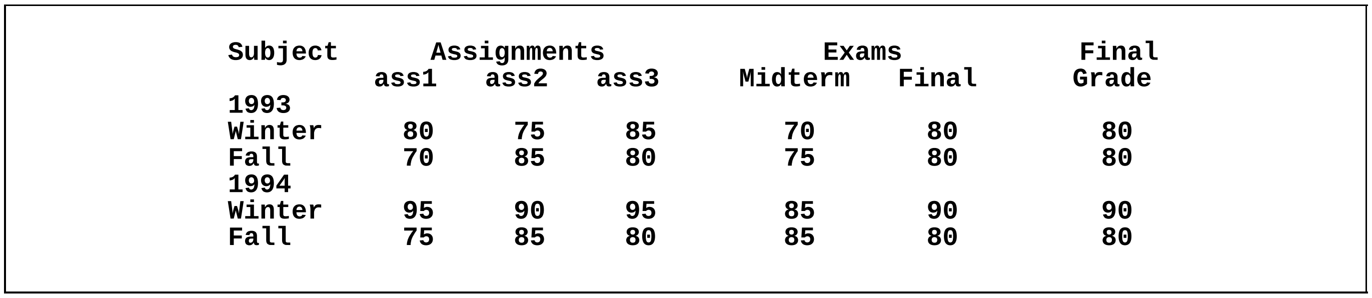


Fearless Forecasters
Economic growth, in percent

	FIRST QUARTER	SECOND QUARTER
Norman Robertson Mellon Bank	4.0	2.3
Neal Soss First Boston	3.1	2.3
Lawrence Kudlow Bear Stearns	3.0	3.5
Donald Ratajczak Georgia State Univ.	3.2	2.8

Figure 4. A table and its row segmentation.

A difficult situation arises when hierarchical row headers are projected onto a single column (Figure 5). In this case, a higher level header (the year "1993" is a qualifier for both "Winter" and "Fall") is undetected and merged with the neighboring lower level header. Since in such layout no spatial boundary is provided between headers at different levels, distinguishing a higher level header from the spill-over of a lower level header is a very difficult task. Solving this problem will again likely involve semantic analysis.



Subject	Assignments			Exams		Final
	ass1	ass2	ass3	Midterm	Final	Grade
1993						
Winter	80	75	85	70	80	80
Fall	70	85	80	75	80	80
1994						
Winter	95	90	95	85	90	90
Fall	75	85	80	85	80	80

Figure 5. A table with hierarchical row headers projected onto a single column.

3. EXPERIMENTAL EVALUATION

In this section, we present our approach to evaluating table structure recognition. We describe in turn the graph model we have adopted, a system we have created for ground-truthing table structure, and our procedure for comparing the output from table recognition to the corresponding ground-truth. The paradigm we have developed is a general one, and could be applied to other algorithms that attempt to extract structure from documents. We also present preliminary experimental results of using our algorithm to recognize two small corpora of tables.

3.1. Graph Model

While it is traditional to regard document analysis results as tree-structured, we have adopted a slightly more general representation, a directed acyclic graph (DAG). This flexibility is important both because there are real-life tables that fall outside the Wang model, and because the output from an imperfect recognition process may not necessarily correspond to a legal instance of a table.

There are two basic classes of nodes in our table DAG: *leaf nodes* which have no children and which contain content corresponding to a specific region on the page (i.e., one or more text strings), and *composite nodes* which are simply unordered collections (sets) of previously-defined leaf and composite nodes. Every node has an optional label. There is, however, no rigid policy enforcing how nodes must be labeled relative to one another. Instead, conventions can be developed on a per-application basis. Indeed, our plan is to apply this same general formalism to other document analysis tasks in the future.

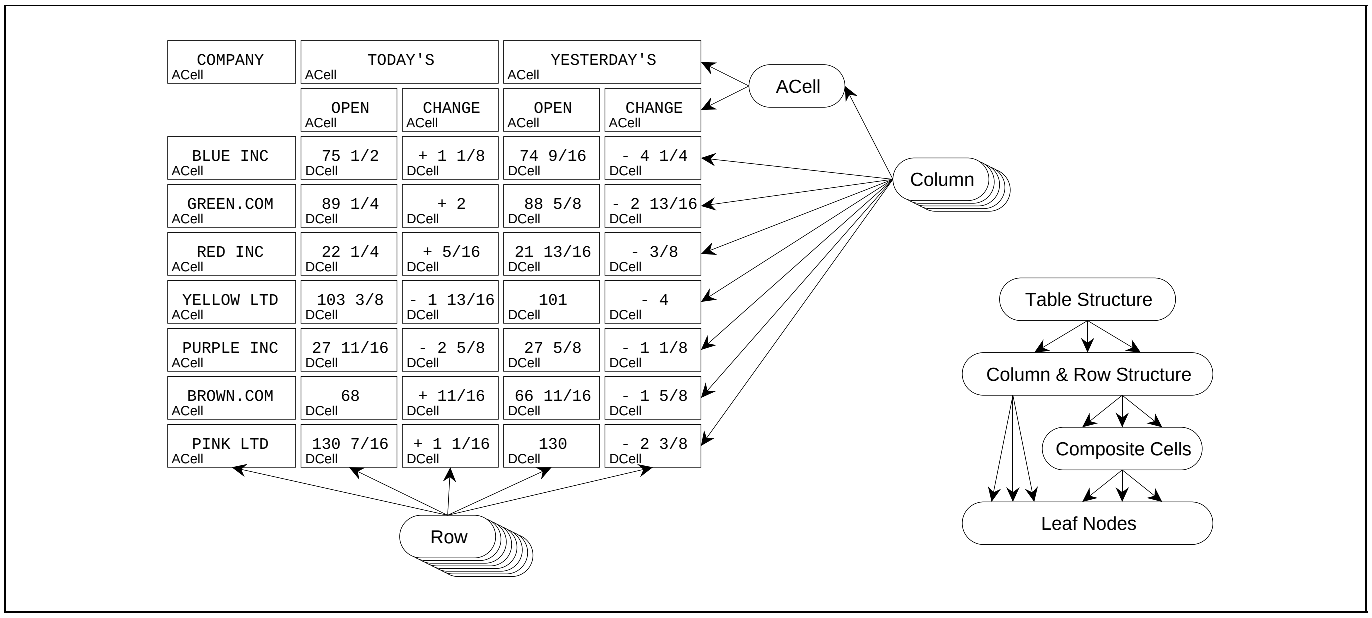
An example of a table DAG is depicted in Figure 6 (to keep the figure comprehensible, we have suppressed many of the edges and nodes present in the full graph). In this graph, there are 28 leaf nodes labeled "DCell" and 14 leaf nodes labeled "ACell," while the node labeled "Column" on the right is a composite node consisting of all the nodes associated with the last column in the table. Note that the bottom-rightmost "DCell" (with content "−2 3/8") is a child both of a node labeled "Row" and of another node labeled "Column;" hence, this graph is not a tree.

3.2. Ground-Truthing

To enable the viewing of document analysis results and to support the ground-truthing process, we have developed an interactive tool we call *Daffy* for browsing and editing table DAG's. The user interface portions of Daffy are written in Tcl/Tk, a powerful and well-known scripting language developed by Ousterhout.[15]

Daffy makes it possible to:

1. display and edit graphical mark-up
2. define new mark-up types
3. examine hierarchical structure
4. print and save PostScript page images
5. run algorithm animation scripts for visualizing the effects of document analysis

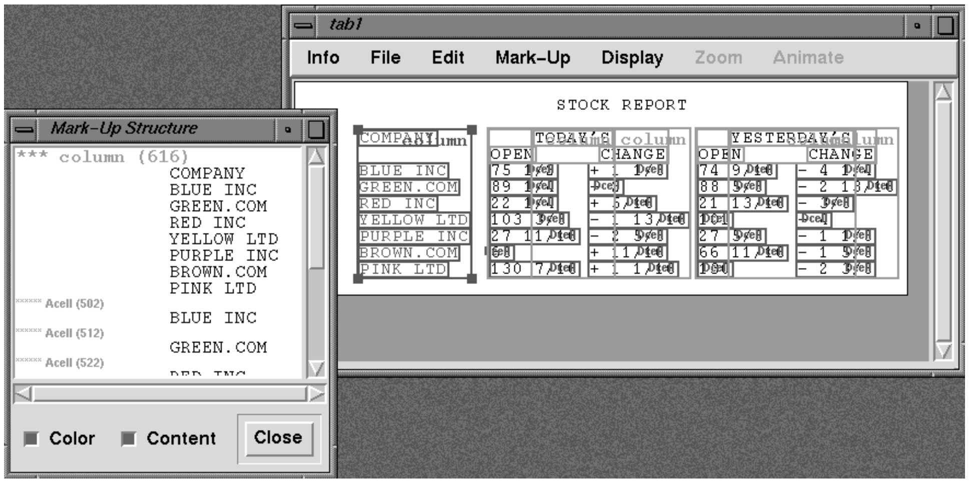


COMPANY ACell	TODAY'S ACell		YESTERDAY'S ACell	
	OPEN ACell	CHANGE ACell	OPEN ACell	CHANGE ACell
BLUE INC ACell	75 1/2 DCell	+ 1 1/8 DCell	74 9/16 DCell	- 4 1/4 DCell
GREEN.COM ACell	89 1/4 DCell	+ 2 DCell	88 5/8 DCell	- 2 13/16 DCell
RED INC ACell	22 1/4 DCell	+ 5/16 DCell	21 13/16 DCell	- 3/8 DCell
YELLOW LTD ACell	103 3/8 DCell	- 1 13/16 DCell	101 DCell	- 4 DCell
PURPLE INC ACell	27 11/16 DCell	- 2 5/8 DCell	27 5/8 DCell	- 1 1/8 DCell
BROWN.COM ACell	68 DCell	+ 11/16 DCell	66 11/16 DCell	- 1 5/8 DCell
PINK LTD ACell	130 7/16 DCell	+ 1 1/16 DCell	130 DCell	- 2 3/8 DCell

Figure 6. Graph representation for table recognition results.

Input is accepted in both image (TIF) and text (ASCII) formats.

Figure 7 presents a screen snapshot of Daffy running on an SGI O2 workstation. The main window, on the right, shows the same sample table document as shown in Figure 1. Several layers of mark-up are visible – on-screen these are displayed in color and are much more legible. Structure corresponding to the leftmost table column which the user has selected is displayed in the child window on the left in the snapshot.



STOCK REPORT

```
*** column (616)
                COMPANY
                BLUE INC
                GREEN.COM
                RED INC
                YELLOW LTD
                PURPLE INC
                BROWN.COM
                PINK LTD
****** Acell (502)
                BLUE INC
****** Acell (512)
                GREEN.COM
****** Acell (522)
```

Figure 7. Daffy screen snapshot.

Daffy supports the full generality of the graph model described in the preceding section. In particular, it manages and updates the table DAG across operations that include adding and deleting leaf nodes, grouping and ungrouping to create composite nodes, moving and resizing nodes, copy-and-pasting, editing type definitions, etc. Consistency of the graph is maintained automatically without placing burdensome restrictions on the user.

3.3. Random Graph Probing

Given the table DAG's for a recognition result and its corresponding ground-truth, it is natural to consider comparing the two as a way of determining how well the algorithm has done. Attempting to compare the graphs directly, however, gives rise to two dilemmas. The first is that a solution would imply a solution to the graph isomorphism problem which is not likely to have an efficient algorithm.[16] While heuristics exist that are sometimes fast, their worst-case behavior is still exponential.[17] Hence, the problem remains a difficult one.

The other obstacle is that there may be several different ways to represent the same table as a graph, all equally applicable. Minor discrepancies in labeling and/or structure could create the appearance that two graphs are dissimilar when in fact they are functionally equivalent from the standpoint of the intended application. Forcing one graph to correspond to the other through a series of rigidly defined editing operations obscures this important point.

At the other end of the spectrum, we could embed our table recognition algorithm in a query-based table processing system,[2] and measure the performance of the complete system on a specific task from a user's perspective: Does it provide the desired information? (this is "goal-directed evaluation" as discussed by Nagy[18]). This approach has its own shortcomings, however, as it limits the generality of the results and makes it difficult to identify the precise source of errors that arise when complex processes interact.

We have developed a third methodology that lies midway between these two which works directly with the graph representation. However, instead of trying to match the graphs under a formal editing model, we probe their structure and content by asking relatively simple queries that mimic, perhaps, the sorts of operations that might arise in a real application.

Conceptually, the idea is to place each of the two graphs under study inside a "black box" capable of evaluating a set of graph-oriented operations (e.g., returning a list of all the leaf nodes, or all nodes labeled in a certain way). We then pose a series of probes and correlate the responses of the two systems. A measure of their similarity is the number of times their outputs agree. Note that it is essential the probes themselves have simple answers that are easily compared. They might return, for example, a count of the number of nodes satisfying a certain property (e.g., possessing a particular label), or the content of a designated leaf node. The probing becomes recursive if the target of a probe is a graph itself (i.e., a composite node). The intention is that this probing process abstracts the access of content away from the specific details of the graph's structural representation.

While the paradigm is open-ended, currently we have defined three categories of probes:

Class 0 These probes count the number of occurrences of a given type of node in the graph. Referring again to Figure 6, a typical Class 0 probe might be paraphrased as: "How many nodes labeled 'Column' does the graph have?" The answer in this case is "5."

Class 1 These probes combine content and label specifications. Currently they apply only to leaf nodes. A representative Class 1 probe might be: "How many leaf nodes labeled 'Acell' with content 'OPEN' does the graph have?" The reply here is "2."

Class 2 These are the most sophisticated probes we have implemented to date. Class 2 probes mimic simple database-type queries, although phrased entirely in terms of graph manipulations. For a given target node, keys that uniquely determine its row and column are identified. These are used to index into the graph, retrieving the content of the node (if any) that lies at their intersection. An example of a Class 2 probe for the graph in Figure 6 is: "What is the value of 'TODAY'S OPEN' for 'RED INC'?" The response would be "22 1/4."

The generation of a probe set is based on one or the other of the graphs in question. That graph will obviously return the definitive responses for all of the probes in the set, while the other graph will do more or less well depending on how closely it matches the first. We then repeat the process from the other direction, generating the probe set from the second graph and tallying the responses for both. The probes are synthesized automatically, working from the table DAG output from the recognition and ground-truthing processes described earlier. For specifying probes, we have implemented a graph-oriented query language embedded in a general-purpose programming language; this offers a great deal of power and flexibility.

3.4. Experiments

The test database was composed of 26 Wall Street Journal articles in text format (WSJ database) and 16 email messages. These were selected by running our table detection algorithm on larger collections.[1] We chose test examples where the output from detection was reasonably good (but not necessarily perfect), passing over particularly "hard" cases where tables were completely missed in the text, split, or merged, or where the human ground-truthers differed significantly in their opinions of the "true" structure of a table. The intention was to examine the performance of table structure recognition across a range of reasonably well-behaved real-world inputs. Each test sample was in a single column format and contained one or more tables. Tables along with the detected boundaries were input to the table recognition algorithm described in Section 2.

As explained in Section 3.3, probing was done bidirectionally, i.e., the recognition result was probed based on its corresponding ground-truth and vice-versa. This involved generating a set of queries to probe the various nodes of the graph which represent the table structure. Three classes of queries were generated in the probing experiment. The agreements of the probes for each of the three classes is plotted in Figure 8 for the WSJ documents and Figure 9 for the email documents. Also superimposed on the plot is the total agreement (combining all the classes). The overall agreement was 82% for the WSJ documents and 73% for the email documents. The better performance for the WSJ database, we believe, is due to the more homogeneous collection of its documents with a few classes of table structures. In sharp contrast, the email documents were a heterogeneous collection with varied layouts.

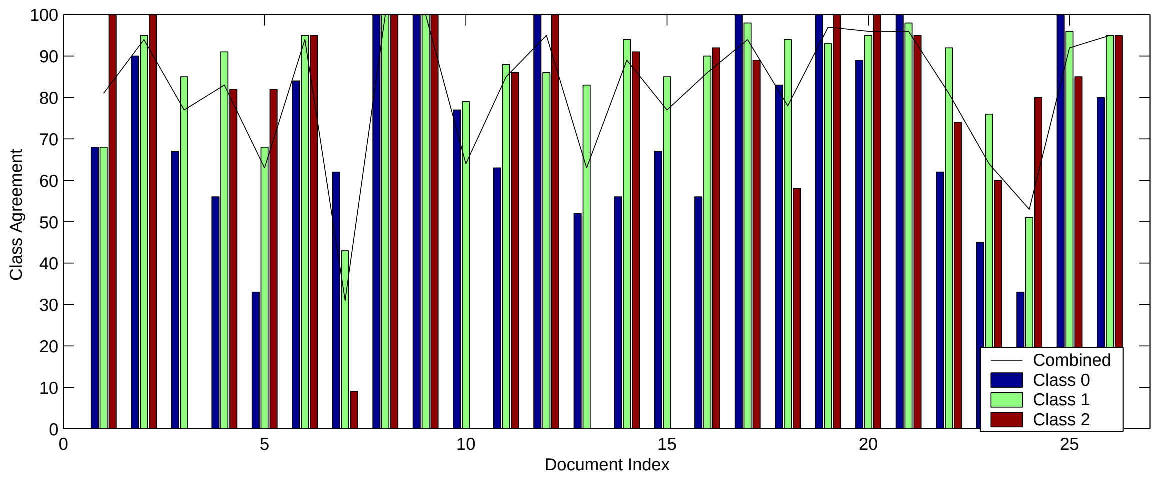


Figure 8. Class agreements for documents in the WSJ database.

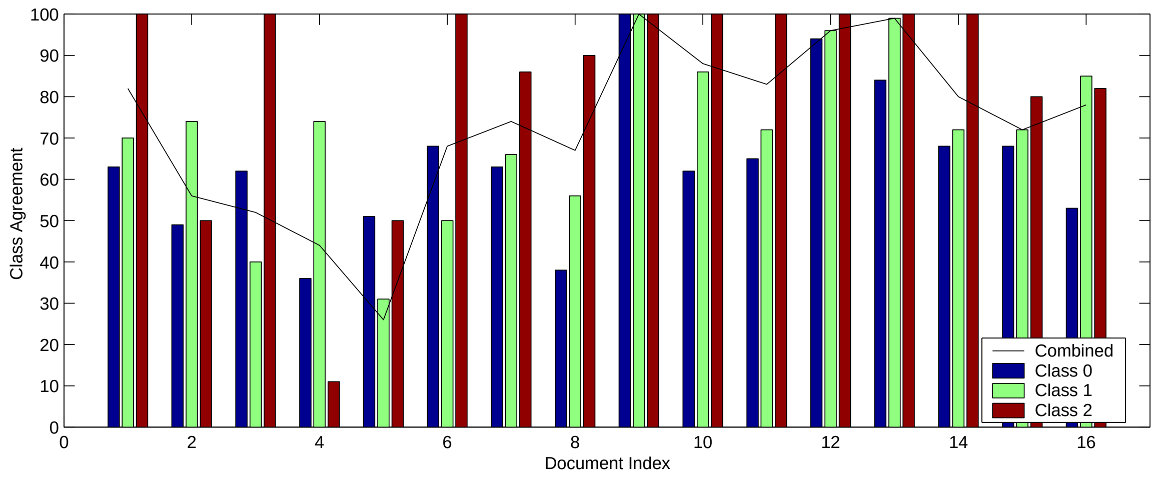
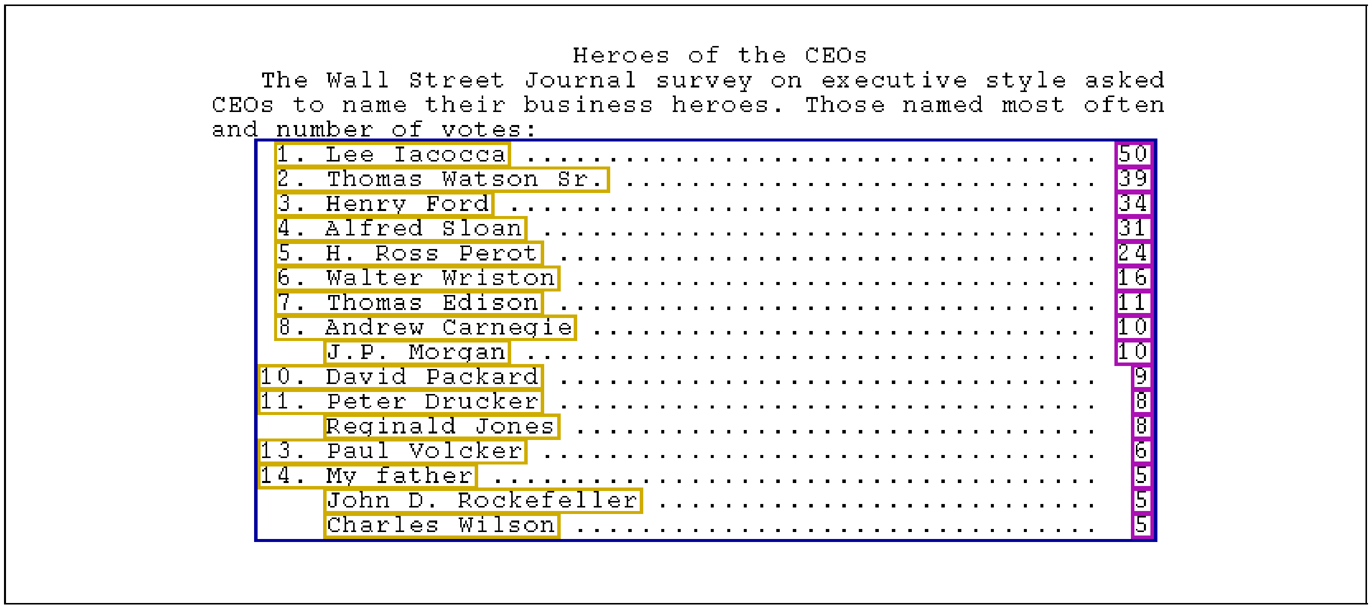


Figure 9. Class agreements for documents in the email database.

Figure 8 contains several documents for which the Class 2 score was zero. One reason was that the table in such documents had only two rows (a row containing headers and a row containing data) and no Class 2 queries were generated. Another reason was that, currently, our algorithm does not capture hierarchical row headers and this leads to incorrect recognition of table structure in such tables. Another cause of low performance is the presence of multiple interpretations of the structure of a table. As seen from Figure 8, the class accuracies are lowest for Document 7. The corresponding document is shown in Figure 10. One plausible interpretation of the table is that it has three columns, with the leftmost column indicating the rank, the center column corresponding to names, and the third column corresponding to the vote count. This is the interpretation adopted in the ground-truth. The recognized result, however, has merged the first two columns as indicated in Figure 10, which represents a different, yet also plausible, interpretation. This difference in the layout structure between the recognition result and the ground-truth leads to its poor performance. The table is complicated further by the fact that a few names share the same rank (the 8th, 11th and 14th). This can be deduced only from a semantic analysis of the text. This example demonstrates that several equally plausible interpretations (ground-truths) of a single table can be made and this makes the evaluation task extremely challenging.



Heroes of the CEOs
The Wall Street Journal survey on executive style asked
CEOs to name their business heroes. Those named most often
and number of votes:

1. Lee Iacocca	50
2. Thomas Watson Sr.	39
3. Henry Ford	34
4. Alfred Sloan	31
5. H. Ross Perot	24
6. Walter Wriston	16
7. Thomas Edison	11
8. Andrew Carnegie	10
J.P. Morgan	10
10. David Packard	9
11. Peter Drucker	8
Reginald Jones	8
13. Paul Volcker	6
14. My father	5
John D. Rockefeller	5
Charles Wilson	5

Figure 10. Screen snapshot of Document 7 in the WSJ database.

As a first step towards correlating our evaluation method based on graph probing with a user's perception of the quality of the recognition results, we conducted an informal experiment. The experiment was performed with three subjects who were not involved in the design of the underlying algorithms. The subjects were shown six test documents (chosen from each of the datasets, labeled A through F for email documents and a through f for WSJ documents), along with their associated ground-truths using the Daffy interface. The six test documents were shown with the table structure marked-up by the recognition algorithm. The corresponding ground-truth was the same document with the table structure marked-up by one of the authors. The task was to rank-order the six documents based on how good a job the algorithm performed at recognizing the table structure regions, as defined by the ground-truth, with rank one corresponding to the best match in the group of six. The subjects were asked only to look at row structure, column structure, Acells and Dcells. They were free to choose any methodology to obtain an overall similarity measure.

The rankings from the three subjects are shown in Table 1 for the email dataset and in Table 2 for the Wall Street dataset. The fourth column represents the computed average ranking obtained by the three subjects. Also shown in the last columns for both the tables are the rankings obtained by using our random graph probing technique to compare the results from the recognition algorithm to the ground-truth. As seen from the tables, the subjects' average ranking compares well with that obtained by our evaluation measure. Individual differences in ranking among subjects are due to the different strategies adopted by each individual to obtain an overall similarity measure.

The larger inconsistency between subjects suggests that it is difficult to judge recognition performance.

Doc Index	Rankings from subjects			Average Ranking	Ranking based on Random Graph Probing
	Sub 1	Sub 2	Sub 3		
A	1	1	1	1	1
B	4	2	2	2	2
C	2	5	5	4	3
D	3	3	6	3	4
E	6	6	3	6	5
F	5	4	4	5	6

Table 1. Ranking from the subjects and our evaluation procedure for the email dataset.

Doc Index	Rankings from subjects			Average Ranking	Ranking based on Random Graph Probing
	Sub 1	Sub 2	Sub 3		
a	1	1	1	1	1
b	2	6	2	3	2
c	3	2	3	2	3
d	6	5	4	6	4
e	4	4	6	5	5
f	5	3	5	4	6

Table 2. Ranking from the subjects and our evaluation procedure for the WSJ dataset.

4. CONCLUSIONS

This paper describes algorithms that recognize tables in ASCII text. Random graph probing was introduced as a new paradigm for evaluating the performance of a table recognition system. Preliminary experiments on ASCII documents demonstrated the effectiveness of the approach. Similar experiments need to be conducted for documents in other media such as scanned images where mechanisms to deal with OCR errors also have to be addressed.

In the area of table understanding, there remain many directions for future work. For example, there is a need to explore more sophisticated table quality measures such as including syntactic and semantic elements of documents both for detection and recognition. Another direction for further study is concerned with the generation of more sophisticated probes for evaluation of table structure and for extending this paradigm to evaluate other algorithms that attempt to extract structure from documents, and more work on correlating this measure with users' perception of the quality of the recognition results.

REFERENCES

1. J. Hu, R. Kashi, D. Lopresti, and G. Wilfong, "Medium-independent table detection," in *Proceedings of Document Recognition and Retrieval VII (IS&T/SPIE Electronic Imaging)*, vol. 3967, pp. 291–302, (San Jose, California), January 2000.
2. J. Hu, R. Kashi, D. Lopresti, and G. Wilfong, "A system for understanding and reformulating tables," in *Proceedings of the 4th IAPR International Workshop on Document Analysis Systems*, (Rio de Janeiro, Brazil), December 2000. To appear.
3. K. Zuyev, "Table image segmentation," in *Proceedings of the Fourth International Conference on Document Analysis and Recognition*, pp. 705–708, (Ulm, Germany), August 1997.
4. E. Green and M. Krishnamoorthy, "Recognition of tables using table grammars," in *Proceedings of the Fifth Annual Symposium on Document Analysis and Information Retrieval*, pp. 261–277, (Las Vegas, Nevada), 1995.
5. C. Peterman and C. Chang, "A system for table understanding," in *Proceedings of the Symposium on Document Image Understanding Technology*, pp. 55–62, (Annapolis, Maryland), 1997.
6. M. Hurst and S. Douglas, "Layout and language: Preliminary investigations in recognizing the structure of tables," in *Proceedings of the Fourth International Conference on Document Analysis and Recognition*, pp. 1043–1047, (Ulm, Germany), 1997.

7. S. Chandran, S. Balasubramanian, T. Gandhi, A. Prasad, and R. Kasturi, "Structure recognition and information extraction from tabular documents," *International Journal of Imaging Systems and Technology* **7**, pp. 289–303, 1996.
8. D. Rus and D. Subramanian, "Customizing information capture and access," *ACM Transactions on Information Systems* **15**, pp. 67–101, January 1997.
9. W. Kornfeld and J. Wattecamps, "Automatically locating, extracting and analyzing tabular data," in *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 347–349, (Melbourne, Australia), 1998.
10. T. G. Kieninger, "Table structure recognition based on robust block segmentation," in *Proceedings of Document Recognition and Retrieval V (IS&T/SPIE Electronic Imaging)*, vol. 3305, pp. 22–32, (San Jose, California), January 1998.
11. H. Ng, C. Y. Lim, and J. Koo, "Learning to recognize tables in free text," in *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pp. 443–450, (Maryland, USA), 1999.
12. X. Wang, *Tabular abstraction, editing, and formatting.* PhD thesis, University of Waterloo, 1996.
13. D. Lopresti and G. Nagy, "Automated table processing: An (opinionated) survey," in *Proceedings of the Third IAPR International Workshop on Graphics Recognition*, pp. 109–134, (Jaipur, India), September 1999.
14. A. Jain and R. C. Dubes, *Algorithms for Clustering Data*, Prentice Hall, 1988.
15. J. K. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley, Reading, MA, 1994.
16. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, San Francisco, CA, 1979.
17. B. T. Messmer and H. Bunke, "Efficient error-tolerant subgraph isomorphism detection," in *Shape, Structure and Pattern Recognition*, D. Dori and A. Bruckstein, eds., pp. 231–240, World Scientific, Singapore, 1995.
18. G. Nagy, "Document image analysis: Automated performance evaluation," in *Document Analysis Systems*, A. L. Spitz and A. Dengel, eds., pp. 137–156, World Scientific, Singapore, 1995.