A System for Understanding and Reformulating Tables

Jianying Hu[1], Ramanujan Kashi[1], Daniel Lopresti[2], and Gordon Wilfong[2]

[1] Avaya Labs, Avaya Communication,
600 Mountain Ave, Murray Hill, NJ 07974, USA
{jianhu,ramanuja}@avaya.com
[2] Bell Labs, Lucent Technologies,
600 Mountain Ave, Murray Hill, NJ 07974, USA
{dpl,gtw}@research.bell-labs.com

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 prototype for a complete, end-to-end table understanding system that takes raw ASCII text as input (*e.g.*, an email message), detects and recognizes the structure of 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. We describe general solutions to the problems of: (1) detecting tables based on optimally partitioning a document into some number of tables using dynamic programming, and (2) recognizing the structure of a detected table region based on hierarchical clustering to identify columns and spatial and lexical criteria to classify headers. A new approach called random graph probing is introduced to evaluate the performance of a table understanding system.

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. Possible applications include extracting information for populating databases which can later be manipulated or queried, and reformulating existing tables so that they can be presented in a medium different than their original target (*e.g.*, on a much smaller screen, or via a spoken language interface).

For example, in a system that reads email over the phone, it would be inconvenient to be forced to listen to the entire body of a large table being read sequentially. On the other hand, many tables are formatted in a way that makes it quite natural to consider querying them via simple relational-database-type commands. Clearly, being able to recognize and exploit such structure in a document would be extremely useful.

In this paper, we describe a prototype for a complete, end-to-end 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. This type of application framework is both compelling and tremendously challenging, as it spans the fields of document analysis, user interface design and development, text-to-speech synthesis, speech recognition, and dialog systems. An overview of our system is shown in Fig. 1. While many of the components are still the subject of active research, preliminary indications are promising.

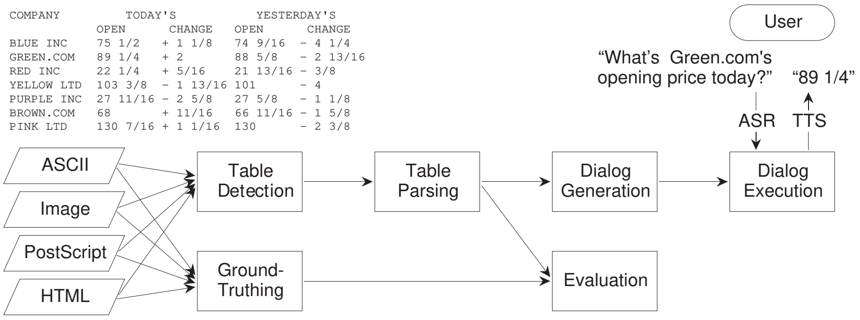

Fig. 1. Overview of the system described in this paper.

The table understanding task can be broken into two logical steps: table detection and table recognition. The first step aims to detect the presence of one or more tables in a document and delineate each table from the surrounding text. The goal of table recognition is to determine the structure (either layout or logical structure) of a given table, and identify functional elements such as columns, rows, headers, etc.

We present separate algorithms for the detection and recognition steps. Both operate without the need to rely on ruling lines, and appear to work reasonably well for "generic" tables. In each case, evidence is collected locally, but decisions are made through an optimal global search, thus improving reliability. Although the current system only takes ASCII text as input, many of the same modules will work for image data (e.g., faxes) as well. In particular, the table detection algorithm has been tested successfully on scanned document images [4]. One unique problem in working with image data is coping with OCR errors, an issue we have not yet addressed.

2 Previous Work

Most prior research on the problem of table detection has concentrated on detecting tables in scanned images, and the vast majority of the work depends on the presence of at least some number of ruling lines (*e.g.*, [9], [3]). Notable exceptions to this assumption include a paper by Rahgozar and Cooperman [13] where a system based on graph-rewriting is described, and one by Shamalian, Baird and Wood [15] in which a system based on predefined layout structures is given.

There is much less prior art in the case of ASCII tables, although these are becoming increasingly important. These may originate either in ASCII form directly (*e.g.*, as part of an e-mail message), or as the result of saving a "richer" document (*e.g.*, an HTML page) in "text-only" format.

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 [17, 2] to segment the table into a structure of cells. Another commonly used technique for detecting columns employs vertical projection profile methods [1, 14]. The work by Hurst and Douglas [5] is concerned with taking a segmented table and using the contents of the resulting cells to determine the logical structure of the table. A method based on $LR(k)$ parsing is mentioned in [8], however it only works for a given class of tables (financial tables). Kieninger [7] proposed a bottom-up approach where vertically overlapping words are grouped into blocks. Various complex heuristics are then applied to split or merge the blocks into proposed columns. Recently, Ng et al. proposed a machine learning based method for both table detection and column/row segmentation [12]. 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.

3 Table Detection Algorithm

The goal of table detection is to detect the presence of one or more tables in a document and delineate their boundaries. We assume that the input is a single column document segmentable into individual, non-overlapping text lines (referred to simply as "lines" henceforth).

While it may be tempting to assume some form of delimiter (*e.g.*, that tables will always be separated from the rest of the text by at least one blank line or some minimum amount of white space), to preserve generality we do not want to make any *a priori* assumptions about where tables might begin or end in the input. Instead, we compute a value for all possible starting and ending positions and then choose the best possible way to partition the input into some number of tables.

Say there are a total of n lines in the input, and let $tab[i, j]$ be a measure of our confidence when lines i through j are interpreted as a single table. Let $merit_{pre}(i, [i+1, j])$ be the merit of prepending line i to the table extending from line $i+1$ to line j, and $merit_{app}([i, j-1], j)$ be the merit of appending line j to the table extending from line i to line $j-1$. We have chosen white space correlation and vertical connected component analysis to model specific functions for $merit_{pre}$ and $merit_{app}$ [4]. As a rule they return larger values for more compatible combinations and can be tuned for specific applications and/or the input medium. Then we define $tab[i,i] = 0,\ 1 \le i \le n$, and

$$tab[i,j] = \max \begin{cases} merit_{pre}(i,[i+1,j]) \ + \ tab[i+1,j] \\ tab[i,j-1] \ + \ merit_{app}([i,j-1],j) \end{cases} \qquad 1 \le i < j \le n \quad (1)$$

This computation builds an upper triangular matrix holding the values for all possible table starting and ending positions.

We then formulate the partitioning of the input into tables as an optimization problem. Let $score[i,j]$ correspond to the best way to interpret lines i through j as some number of (*i.e.*, zero or more) tables. The computation is defined as $score[i,i] = tab[i,i],\ \ 1 \le i \le n$, and

$$score[i,j] = \max \begin{cases} tab[i,j] \\ \max_{i \le k < j} \{score[i,k] \ + \ score[k+1,j]\} \end{cases} \ 1 \le i < j \le n \ \ (2)$$

The precise decomposition of a page can be obtained by backtracking the sequence of decisions made in computing $score[1,n]$. Any region on the optimal path whose tab value is higher than a predetermined threshold is considered a table region. The final output of the table detection stage is the boundaries of the detected tables.

4 Table Structure Recognition

Many different terminologies have been used by researchers to describe the structure of a table. We chose to base our model on Wang's formalism [16] 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.

Fig. 2 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. 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. For the graph corresponding to the table depicted in Fig. 2, there are 28 leaf nodes labeled *Dcell* and 14 leaf nodes labeled *ACell*,

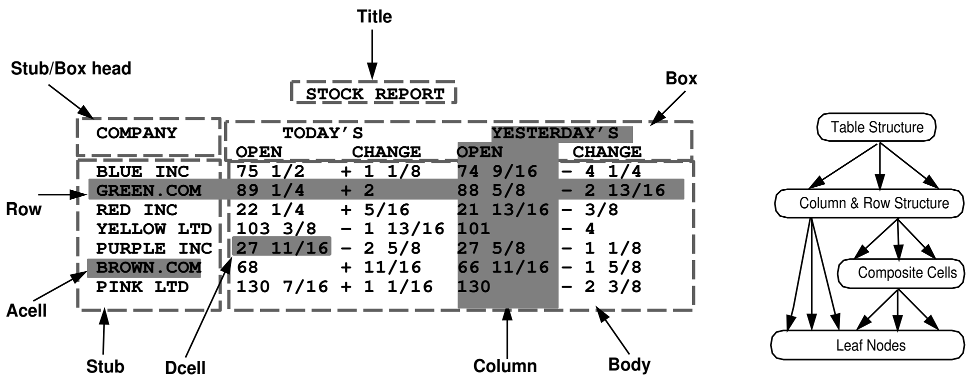

Fig. 2. Table terminology (adapted from Wang's Ph.D. thesis) [16].

while *Row* and *Column* are composite nodes. Note that the indicated *DCell* (with content "27 11/16") is a child both of a node labeled *Row* (headed by "PURPLE INC") and of another node labeled *Column* (headed by "TODAY'S OPEN"); hence, this graph is not a tree.

4.1 Column Segmentation

The input to the column segmentation step is the boundaries of 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. 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 constructed in a bottom-up manner [6]. 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. The merging process is repeated recursively until there is only one cluster left. The nature of the hierarchical clustering algorithm insures that the proposed column segmentation algorithm can handle imperfect vertical alignment very well.

In this application, each word w_i is represented by its starting and ending horizontal positions encoded by the position vector $p_i = (s_i, e_i)$, also called the *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. 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. The cut where each resulting cluster corresponds to a column is found using a breadth-first traversal of the cluster tree starting from the root. A set of heuristics is applied to determine how far to go along each branch to avoid splitting (going too far) or merging (not going far enough) columns. The main heuristic currently used is that the spacing between table columns tends to be more or less even across the table.

4.2 Header Detection and Row Segmentation

The potential headers are identified using a lexical distance measure and assuming typical layout rules for headers used in most tables. 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. Fig. 3 shows the box of the table in Fig. 2 represented as a tree. This tree is then traversed to assign headers to each column (higher lever headers are shared by more than one column).

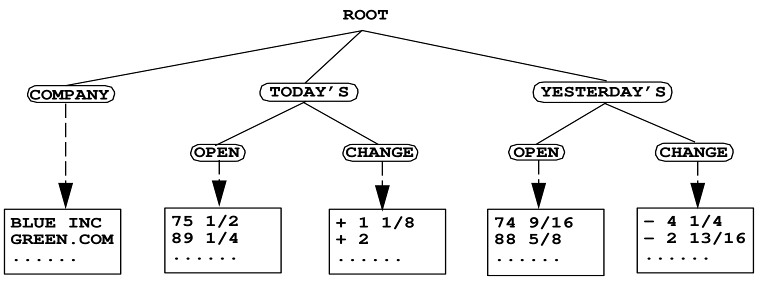

Fig. 3. The tree representation of the box of the table in Fig. 2.

Row segmentation is currently carried out using some simple heuristics: (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. More sophisticated methods for line grouping using statistical syntax analysis such as N-grams will likely improve the performance and will be investigated in our future work.

5 Evaluation via Random Graph Probing

In this section we introduce a new approach to evaluating the performance of a table recognition system. The method is based on random graph probing, and is general enough that it could potentially be used to evaluate any algorithm that attempts to extract certain structures from a document.

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. Daffy makes it possible to: (1) display and edit graphical mark-up; (2) define new mark-up types; (3) examine hierarchical structure; and (4) run algorithm animation scripts for visualizing the effects of document analysis. Daffy supports the full generality of the graph model mentioned in the preceding section for both image (TIF) and text (ASCII) input.

Given the table DAG from 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. However, any reasonable notion of graph matching subsumes the subgraph isomorphism problem which is NP-complete. While heuristics exist that are sometimes fast, their worst-case behavior is still exponential (see, *e.g.*, [10]). Another 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.

At the other end of the spectrum, we could integrate table recognition with the rest of the modules outlined in the Introduction 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 in [11]). 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 (such as those in Fig. 1) interact.

Our methodology lies in between these two extremes. We work 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, each of the two graphs under study is placed 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 Fig. 2, 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 table in Fig. 2 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.

6 Experimental Results

In this section, we present preliminary experimental results of using our algorithm to detect and recognize tables in ASCII text documents. The test database was composed of twenty six Wall Street Journal (WSJ) articles and twenty five email messages. Each test sample was in single column format and contained one or more tables.

6.1 Table Detection

In considering the output from our table detection algorithm, it becomes evident that detection errors can include non-table regions improperly labeled as tables (insertion errors), tables missed completely (deletion errors), larger tables broken into a number of smaller ones (splitting errors), and groups of smaller tables combined to form larger ones (merging errors). This leads naturally to the use of an edit distance model for assessing the results of table detection [4].

A one-to-one matching was done on the test sample and its corresponding ground truth. The ground truth was generated by manually delineating the boundaries of the tables. A useful feature of this evaluation scheme is that it performs error analysis by quantifying the types of detection errors. To illustrate this point, the sum of the errors due to insertion, deletion, substitution, merges and splits for the documents of each test corpus was plotted against the threshold, the value above which a region in a document is considered a table. This is shown for both the WSJ and email documents in Fig. 4. As can be seen, in Fig. 4, for both sets of documents, errors at lower threshold values (left portions of the individual plots) are likely to be insertion errors, which drop as the threshold is increased. On the other hand, at higher threshold values, deletion errors increase significantly. Substitution errors are tables which have been detected but not accurately delineated. This is partly due to the header lines which have been missed by the table detection algorithm. The drop-off in substitution errors in both the corpora at higher thresholds are due to the fact that entire tables were missed at these thresholds and these were classified as deletion errors.

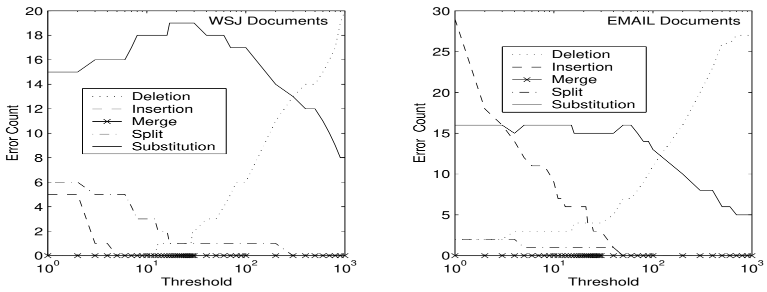

Fig. 4. Count of detection errors for WSJ and EMAIL documents.

6.2 Table Recognition

For the recognition evaluation, tables that were split or merged by our table detection algorithm were removed from further consideration. All the remaining tables along with the detected boundaries were input to the table recognition algorithm described in Section 4. The functional elements which include headers, rows and columns are represented as a graph. As explained in Section 5, random graph probing was used to evaluate the performance of the table recognition algorithm. The test documents used for table recognition were ground-truthed manually using the Daffy interface.

The net agreement of the probes for each of the three classes is plotted in Fig. 5 for the WSJ documents, and in Fig. 6 for the email documents. Also superimposed on the plots is the total accuracy (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 its more homogeneous nature of its documents with only a few basic styles of table layout. In sharp contrast, the email documents were a heterogeneous collection with varied layouts.

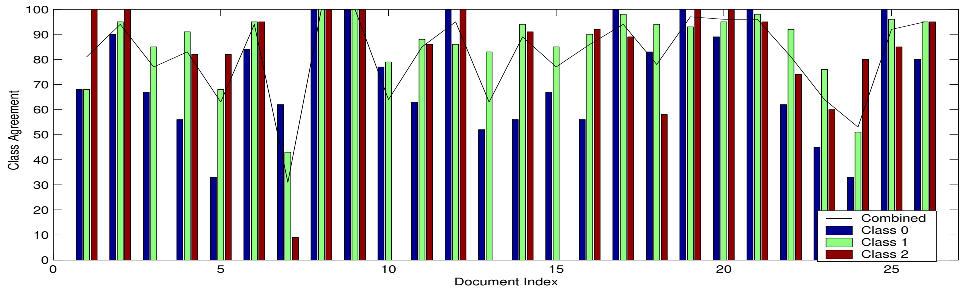

Fig. 5. Class agreement for documents in the WSJ database.

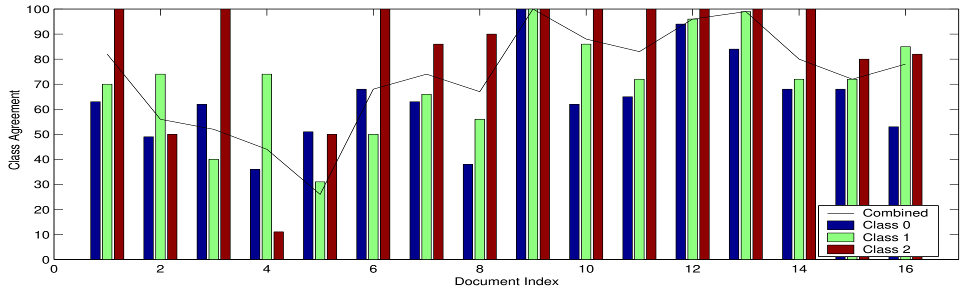

Fig. 6. Class agreement for documents in the email database.

Fig. 5 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. It must also be noted that several plausible interpretations of a single table (ground-truths) can be made and this makes the evaluation task extremely challenging.

7 Table Query Interface

For mobile and remote information access, it is desirable to be able to reformulate a recognized table to facilitate query by a spoken language interface. To enable this and other potential applications we have developed a prototype platform, *MEdit*, for editing, debugging, and demonstrating structured man-machine dialogs. It is intended to be easy enough to use that non-specialists can quickly author dialogs with little or no programming effort, yet powerful enough that sophisticated dialogs can be supported. The system makes extensive use of Bell Labs speech recognition (ASR) and text-to-speech (TTS) technologies.

Like Daffy, MEdit is interactive and employs a graphical user interface programmed in Tcl/Tk. At the highest level, MEdit adopts a finite state machine

(FSM) paradigm. The mouse is used to place and move nodes, and to draw edges to connect them. Execution of the dialog can be initiated at any selected node, or at a single designated "start" node. At a slightly lower level, each node in the FSM corresponds to an arbitrary program (*i.e.*, script) written in Tcl. This program is executed whenever the node in question is entered. Edges correspond to two programs: one called when the edge is evaluated by the node it leads from, and another called when the edge is traversed.

When nodes and edges are instantiated, they are given default scripts. Complete, simple dialogs can be built by replacing one or more values in these templates. The user is in no way limited by the default behavior of nodes and edges, however, and arbitrarily complicated specifications can be written and executed. In addition to allowing interactive dialog construction, MEdit provides an interface for building dialogs programmatically; programs (*e.g.*, a table recognition system) can write programs that generate dialogs.

The following transcript illustrates a session involving our table example:

SYSTEM: *I found a table with seven rows and five columns. The column headings are: company, today's open, today's change, yesterday's open, and yesterday's change. What next?*

USER: *Read company.*

SYSTEM: *Column one, company, is: Blue Inc, Green.com, Red Inc, Yellow Ltd, Purple Inc, Brown.com, and Pink Ltd. What next?*

USER: *What is today's open for Green.com?*

SYSTEM: *Today's open for Green.com is* 89 1/4. *What next?*

USER: *Read Purple Inc.*

SYSTEM: *Purple Inc, today's open* 27 11/16, *today's change* −2 5/8, *yesterday's open* 27 5/8, *yesterday's change* −1 1/8. *What next? ...*

While still far from human-like, it is clear how such an access mechanism could greatly assist a user attempting to retrieve information from a document over the phone.

8 Conclusions

This paper has presented a system and its associated algorithms that detects and recognizes tables in ASCII text. Random graph probing was introduced as a new paradigm for evaluating the performance of a table recognition system. Further, a prototype was described which generates a simple interactive dialog to query the recognized table using a spoken language interface. Preliminary experiments on ASCII documents demonstrated the effectiveness of the approach. Future work involves exploring more sophisticated table quality measures such as making use of syntax and semantics both for detection and recognition. Another area for further study is in the generation of more sophisticated probes for evaluating table structure, and extending this paradigm to other algorithms that attempt to extract structure from documents.

References

1. 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*, 7:289–303, 1996.
2. E. Green and M. Krishnamoorthy. Recognition of tables using table grammars. In *Proceedings of the 5th Annual Symposium on Document Analysis and Information Retrieval*, pages 261–277, Las Vegas, Nevada, 1995.
3. Y. Hirayama. A method for table structure analysis using DP matching. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, pages 583–586, Montréal, Canada, August 1995.
4. J. Hu, R. Kashi, D. Lopresti, and G. Wilfong. Medium-independent table detection. In *Proceedings of IS&T/SPIE Document Recognition and Retrieval VII*, pages 291–302, San Jose, California, January 2000.
5. M. Hurst and S. Douglas. Layout and language: Preliminary investigations in recognizing the structure of tables. In *Proceedings of 4th International Conference on Document Analysis and Recognition*, pages 1043–1047, Ulm, Germany, 1997.
6. A. Jain and R. C. Dubes. *Algorithms for Clustering Data.* Prentice Hall, 1988.
7. T. G. Kieninger. Table structure recognition based on robust block segmentation. In *Proceedings of IS&T/SPIE Document Recognition and Retrieval V*, volume 3305, pages 22–32, San Jose, California, January 1998.
8. 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*, pages 347–349, Melbourne, Australia, 1998.
9. A. Laurentini and P. Viada. Indentifying and understanding tabular material in compound documents. In *Proceedings of the11th International Conference on Pattern Recognition*, pages 405–409, The Hague, The Netherlands, 1992.
10. B. T. Messmer and H. Bunke. Efficient error-tolerant subgraph isomorphism detection. In D. Dori and A. Bruckstein, editors, *Shape, Structure and Pattern Recognition*, pages 231–240. World Scientific, Singapore, 1995.
11. G. Nagy. Document image analysis: Automated performance evaluation. In A. L. Spitz and A. Dengel, editors, *Document Analysis Systems*, pages 137–156. World Scientific, Singapore, 1995.
12. H. Ng, C. Y. Lim, and J. Koo. Learning to recognize tables in free text. In *Proc. 37th Annual Meeting of the Association for Computational Linguistics*, pages 443–450, Maryland, USA, 1999.
13. M. Rahgozar and R. Cooperman. A graph-based table recognition system. In *Proceedings of IS&T/SPIE Document Recognition and Retrieval III*, pages 192–203, San Jose, California, 1996.
14. D. Rus and D. Subramanian. Customizing information capture and access. *ACM Transactions on Information Systems*, 15(1):67–101, January 1997.
15. J. H. Shamalian, H. S. Baird, and T. L. Wood. A retargetable table reader. In *Proceedings of 4th International Conference on Document Analysis and Recognition*, pages 158–163, Ulm, Germany, August 1997.
16. X. Wang. *Tabular abstraction, editing, and formatting.* PhD thesis, University of Waterloo, 1996.
17. K. Zuyev. Table image segmentation. In *Proceedings of 4th International Conference on Document Analysis and Recognition*, pages 705–708, Ulm, Germany, August 1997.