Medium-Independent Table Detection *

Jianying Hu Ram Kashi Daniel Lopresti Gordon Wilfong

Bell Laboratories, Lucent Technologies, Inc.
600 Mountain Avenue, Murray Hill, NJ 07974
{jianhu,ramanuja,dpl,gtw}@research.bell-labs.com

ABSTRACT

An important step towards the goal of table understanding is a method for reliable table detection. This paper describes a general solution for detecting tables based on computing an optimal partitioning of a document into some number of tables. A dynamic programming algorithm is given to solve the resulting optimization problem. This high-level framework is independent of any particular table quality measure and independent of the document medium. Moreover, it does not rely on the presence of ruling lines or other table delimiters. We also present table quality measures based on white space correlation and vertical connected component analysis. These measures can be applied equally well to ASCII text and scanned images.

We report on some preliminary experiments using this method to detect tables in both ASCII text and scanned images, yielding promising results. We present detailed evaluation of these results using three different criteria which by themselves pose interesting research questions.

Keywords: table detection, document layout analysis, connected component analysis, document understanding, information retrieval, multimedia communications .

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 reformatting 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).

From the standpoint of the expected input, these applications can be partitioned into two distinct categories: large-scale processing of identically structured tables in a predetermined format (*e.g.*, phone company billing statements), and more modest-scale processing of tables in widely ranging formats (*e.g.*, tables found during Web searches, tables in faxes or in personal e-mail). The latter category is particularly challenging because of the variety of styles that must be handled "on-the-fly," and by the fact that there is no guarantee the input table will be "well-formed".

The table understanding problem can be broken into two logical steps: table detection and table recognition. Much existing work on tables described in the literature addresses the latter step and assumes that the table has already been identified and segmented out from the input (or that identifying the table is trivial – *e.g.*, the whole document is the table). The goal of table recognition includes determining the structure (either layout or logical structure) of the given table. 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[1,2] to segment the table into a structure of cells. Others also use OCR results to aid in the segmentation of the table into regions such as *body* (that is, the actual cells containing the tabular information), *title block*, *column headers* and *row labels*.[3] The work by Hurst and Douglas[4] is concerned with taking a segmented table and using the contents of the resulting cells to determine the logical structure of the table.

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.*, the work by Laurentini and Viada[5]). Hirayama[6] uses ruling lines as initial evidence of a table or figure and then further refines this decision to distinguish tables from figures by a measure based on such features as the presence of characters. There

* To be presented at SPIE Document Recognition and Retrieval VII, 23 – 28 January 2000, San Jose, California, USA.

is, of course, no guarantee that such lines will be present in printed tables. Notable exceptions to this assumption include a paper by Rahgozar and Cooperman[7] where a system based on graph-rewriting is described, and one by Shamalian, Baird and Wood[8] 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. More often than not, ASCII tables contain no ruling lines whatsoever, depending only on the 2-D layout of the cell contents to convey the table's structure. Hence, very little of the past research on printed tables is applicable in this case. Kieninger[9] describes a system for parsing tables in ASCII or paper documents that does not rely on ruling lines. This is based on a straightforward clustering technique followed by a number of useful post-processing heuristics. A method based on $LR(k)$ parsing is mentioned in[10] for a given class of tables (financial tables). However, neither of these works explicitly considers the detection of tables.

In this paper, we describe a technique for detecting tables that does not rely on ruling lines and has the desirable property that an identical high-level approach can be applied to tables expressed as ASCII text and those in image format. The remainder of this paper is organized as follows. Section 2 describes the high-level structures of an algorithm for detecting any number of tables in a document. The general algorithm is stated in terms independent of any particular table quality measure. Two possible table quality measures are then presented. Section 3 presents our experimental setup and results for both scanned document images as well as ASCII based documents. Finally, conclusion and future directions are given in Section 4.

2. ALGORITHMS

In this section we describe our approach to solving the table detection problem. It consists of: (1) a high-level framework that determines the optimization problem and an algorithm for its solution, and (2) table quality measures that can be tuned for specific applications and/or the input media. We assume that the input is a single column document segmentable into individual, non-overlapping text lines (referred to simply as "lines" henceforth). Our general framework is independent of the table quality measures and, in particular, independent of the input medium. It may in fact be desirable to experiment with a number of different table quality measures to determine which works best in a given situation.

2.1. High-Level Framework

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 table(s) 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 $j-1$. Specific functions for $merit_{pre}$ and $merit_{app}$ will be described in the next subsection, but as a rule they return larger values for more compatible combinations. Then we define:

$$tab[i, i] = 0 \qquad 1 \leq i \leq n \tag{1}$$

and

$$tab[i, j] = \max \begin{cases} merit_{pre}(i, [i+1, j]) \ + \ tab[i+1, j] \\ tab[i, j-1] \ + \ merit_{app}([i, j-1], j) \end{cases} \qquad 1 \leq i < j \leq n \tag{2}$$

This computation builds an upper triangular matrix holding the values for all possible table starting and ending positions. This requires time $O(n^2)$.

The best (*i.e.*, highest quality) table in the input can then be found by searching the $tab[i, j]$ matrix for its maximum value. Say this occurs at index $[a, b]$. This means the best table extends from line a to line b. The second-best table can then be located by excluding the region $[a, b]$ and searching the regions $[1, a-1]$ and $[b+1, n]$ for the next highest value. This is depicted in the diagram on the left side of Figure 1. For any $k > 0$, we can continue this

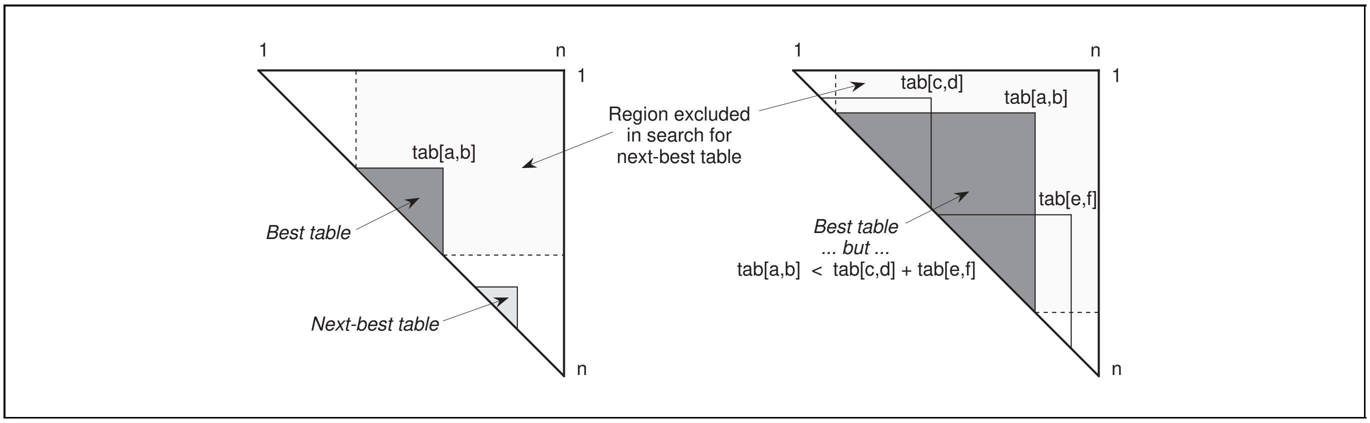


Figure 1. Partitioning the input into its best interpretation as some number of tables.

"greedy" procedure until we have found the k best tables (or, more likely, until the table quality measure has fallen below some predetermined threshold).

This approach could, however, lead to an unsatisfactory solution when the document contains more than one table. Consider the situation illustrated on the right side of Figure 1. Here the region corresponding to $tab[a, b]$ is the single best table in the document. There are, however, two other tables, $tab[c, d]$ and $tab[e, f]$, whose combined confidence scores are greater than that of $tab[a, b]$ alone. We are excluded from choosing these tables, however, because they overlap with $tab[a, b]$ which was selected first.

This situation can be remedied by formulating 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 follows:

$$score[i, i] = tab[i, i] \qquad 1 \leq i \leq n \tag{3}$$

and

$$score[i, j] = \max \begin{cases} tab[i, j] \\ \max_{i \leq k < j} \{score[i, k] \; + \; score[k + 1, j]\} \end{cases} \qquad 1 \leq i < j \leq n \tag{4}$$

One way to interpret Equation 4 is that the best way to partition lines i through j into some number of tables is to:

1. treat lines i through j as coming from a single table, or
2. break the region into two subregions between lines k and $k + 1$ and consider each separately.

Note that if region $[i, j]$ contains exactly one table (or a portion thereof), case (1) will apply. Otherwise, there must surely exist a k such that the region can be broken into two independent subregions (without splitting a table) and then case (2) applies. Because of the two levels of optimization, algorithm *score* requires time $O(n^3)$.

The computation defined by *score* (Equation 4) is somewhat different from that defined by *tab* (Equation 2). Whereas $tab[i, j]$ represents the quality of the region covering lines i through j when interpreted as a *single table*, $score[i, j]$ represents the best way to decompose this region into *some number of tables* (*i.e.*, zero or more) along with separate, non-table text lines. The precise decomposition can be obtained by backtracking the sequence of decisions made in evaluating Equation 4; this gives us the globally optimal strategy for partitioning the input into however many tables it may have contained. The *score* algorithm correctly selects $tab[c, d]$ and $tab[e, f]$ in the example on the right side of Figure 1.

2.2. Specific Table Quality Measures

One of the simplest quality measures imaginable is the degree to which the white space in the text line to be added correlates to the inter-column spacing in the presumed table. On a given line, we distinguish between *inside spaces* and *outside spaces*; the former have at least one non-space character to the left and to the right, while the latter have no non-space characters on one or both sides.

In the case of ASCII input, let α and β correspond to specific character positions on two lines. We define:

$$acorr(\alpha,\beta) = \begin{cases} 1 & \text{if } \alpha \text{ and } \beta \text{ are both inside space characters} \\ -1 & \text{if exactly one of } \alpha \text{ or } \beta \text{ is an inside space} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

Let $atext[i,k]$ be the input text where the first index designates the line and the second the character on that line. For two lines i and j, we pad the shorter line with (outside) spaces to match the length of the longer line. Let this length be m. Now we define the first line correlation measure:

$$lncorr_{ws}(i,j) = \sum_{k=1}^{m} acorr(atext[i,k], atext[j,k]) \tag{6}$$

This is a measure of the correlation between the white space "streams" on the two lines in question. A positive value indicates the lines correlate well (*i.e.*, they may potentially belong to the same table). Figure 2 illustrates these computations for two text lines that correlate well (top) and badly (bottom).

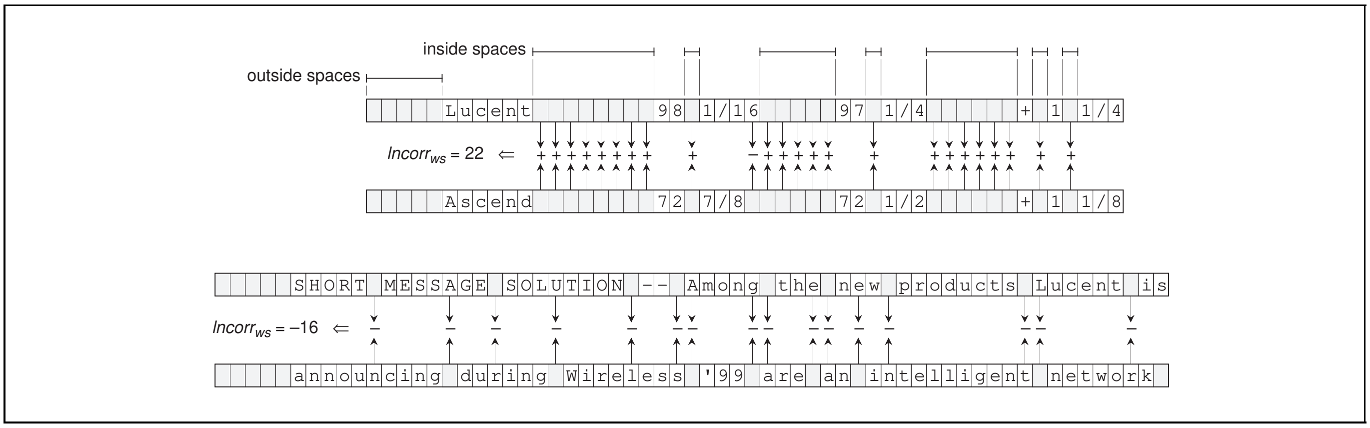


Figure 2. Illustrations of white space correlations for ASCII input.

We now define the set of merit functions:

$$merit_{pre}(i,[i+1,j]) = \sum_{k=i+1}^{j} \frac{1}{e^{\gamma(k-i-1)}} \cdot lncorr_{ws}(i,k) \tag{7}$$

and

$$merit_{app}([i,j-1],j) = \sum_{k=i}^{j-1} \frac{1}{e^{\gamma(j-1-k)}} \cdot lncorr_{ws}(k,j) \tag{8}$$

where γ is a constant that determines the exponential decay. This gives one possible criterion for judging the quality of ASCII tables. Figure 3 shows an example of a table detection result for ASCII input.

Note that the definition of *acorr* can be made much more sophisticated; it could, for example, also provide a "bonus" for correlating two numeric characters (another strong table indicator). Likewise, $lncorr_{ws}$ could compute a line correlation based on, say, edit distance (allowing character insertions and deletions) as opposed to this simple Hamming-type distance.

In the case of image input, we can define an analogous measure computed from word bounding boxes, obtained from low level layout analysis.[11] For this discussion it will be convenient to treat each segmented text line in the image as a string of pixel positions in raster order. Let α and β correspond to specific pixel positions along the length of two lines. We define:

$$icorr_{ws}(\alpha,\beta) = \begin{cases} 1 & \text{if } \alpha \text{ and } \beta \text{ are both inside space pixels} \\ -1 & \text{if exactly one of } \alpha \text{ or } \beta \text{ is an inside space} \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

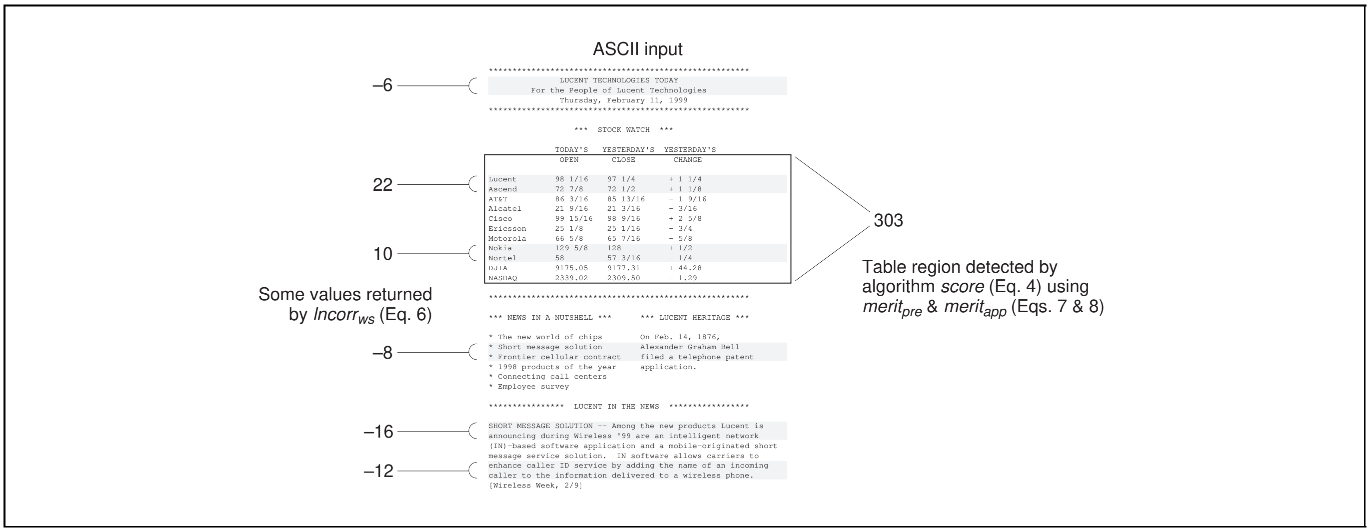


Figure 3. ASCII table detection example.

Mirroring the ASCII case, let $itext[i,k]$ be the image input segmented into text lines where the first index designates the line and the second the pixel position on that line. Equation 6 for $lncorr_{ws}$ is then re-written in terms of $itext$ and the pixel-based white space correlation defined by Equation 9, where m now represents the length in pixels of the longer of the two lines.† Figure 4 illustrates this; note the similarities to the topmost example from Figure 2. The two merit functions (*i.e.*, Equations 7 and 8) are then specified exactly as before.

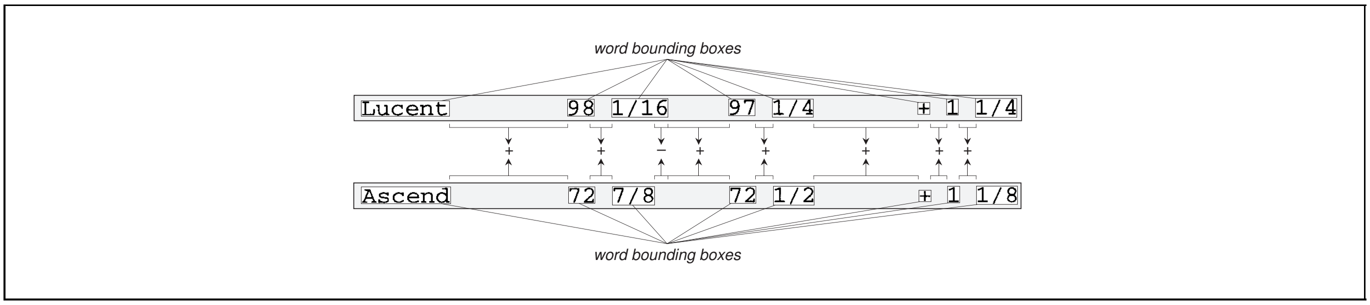


Figure 4. Illustration of white space correlations for image input.

Another possible way of formulating merit functions is through vertical connected component analysis (VCCA). The basic units in VCCA are words. Words in ASCII text are sequences of characters delineated by spaces. In printed text, word bounding boxes can be derived from low-level layout analysis.[11] For this application, the vertical "connectedness" is defined in the following way. Suppose the horizontal extents of two words W_1 and W_2 on adjacent lines are (x_{11}, x_{12}) and (x_{21}, x_{22}) respectively. Assuming $x_{11} \leq x_{21}$ without loss of generality, W_1, W_2 are vertically connected if and only if:

$$\frac{x_{12} - x_{21}}{min((x_{12} - x_{11}), (x_{22} - x_{21}))} > \delta \tag{10}$$

where δ is a threshold (*e.g.*, 0.6). Intuitively, two words on adjacent lines are considered vertically connected if they overlap significantly and have similar lengths. The transitive closure of all pairs of vertically connected words defines a vertical connected component. The connected components can be efficiently computed using an equivalence class algorithm with complexity $O(M+N)$, where M is the number of words and N is the number of vertically connected

†This pixel-by-pixel summation is a convenient way of illustrating the computation. In an actual implementation, however, $lncorr_{ws}$ is computed much more efficiently using the starting and ending coordinates of the word bounding boxes along the two lines.

pairs of words.[12] Kieninger used a similar operation for "block" expansion, but with no constraints on the widths of the overlapping words and the overlap itself.[9]

One problem with the above algorithm is that the connected components thus defined are broken up by occasional "inconsistent" lines, *i.e.*, a blank line or a line of run-on text. These lines are often caused by mistakes in typing (in ASCII tables) or errors in low-level layout analysis (in scanned tables), but can also be inserted intentionally. Human eyes can easily detect the continuity of connected components across such disruptions.

One way to capture such breaks in continuity is to fill the gaps and delete the protruding lines using morphological dilation and erosion operations before VCCA is applied.[13] However our experiments showed that these operations tend to have many unwanted side effects which lead to more false positives in table detection. Therefore we modify the VCCA process itself instead. In particular, when generating pairs of words that are vertically connected, instead of looking only at words on adjacent lines, we examine all words within K lines of each other, where K is the maximum length of the break (currently set to 2). This may lead to unconventional connected components that contain holes or a piece of a different component, but it works well for the purpose of table detection.

Vertical connected components roughly correspond to the columns of a table. The "connectedness" criterion (Equation 10) is designed such that it insures some amount of consistency within a column while allowing for skewed (non-Manhattan) columns. Although at this stage connected components are by no means perfect columns, they capture crucial structural information which can be used to measure the consistency between a line and a region. Figure 5 shows a portion of an ASCII table on the top and the corresponding connected components (with different upper-case characters representing different components) on the bottom.

```
18      Joe Olive  2.240 -0.314 -0.801  0.480 1.604
19   Dan Lopresti  2.240 -0.314  0.636 -0.381 2.181
20 Ramanujan Kashi  2.240 -0.314 -0.801  1.340 2.464

21 Gordon Wilfong -0.439 -0.314  2.073  1.340 2.660
22    Jianying Hu -0.439 -0.314  2.073  1.340 2.660
```

```
AA      BBB CCCCC  DDDDD EEEEEE FFFFFF  GGGGG HHHHH
AA   III CCCCCCCC  DDDDD EEEEEE  FFFFF GGGGGG HHHHH
AA JJJJJJJJJ CCCCC  DDDDD EEEEEE FFFFFF  GGGGG HHHHH

AA JJJJJJ CCCCCCC DDDDDD EEEEEE  FFFFF  GGGGG HHHHH
AA    JJJJJJJJ KK DDDDDD EEEEEE  FFFFF  GGGGG HHHHH
```

Figure 5. ASCII text (top) and the corresponding connected components (bottom).

After VCCA, each line is assigned a list of records containing the labels (and any other necessary attributes) of connected components intersecting this line. For any pair of lines i and j, let $S(i,j)$ denote the set of connected components intersecting both lines, called *shared components*; and $U(i,j)$ denote the set of connected components intersecting only one of the lines, called *unique components*. We now define our second line correlation measure:

$$lncorr_{cc}(i,j) = \sum_{k \in S(i,j)} SC_s(k) + \sum_{k \in U(i,j)} SC_u(k) \tag{11}$$

where $SC_s(k)$ and $SC_u(k)$ are scores assigned to connected component k depending on whether it is a shared or unique component. These scores could in general be functions of the attributes (*e.g.*, height) of the connected component, but in our initial investigations they are assigned constant values (2.5 and -1.0 respectively). Note that although this second line correlation measure seems to depend only on two neighboring lines, it in fact incorporates much more global information through the precomputed connected components.

The two line correlation measures (Equations 6 and 11) are complementary to each other in the sense that the first measure is more localized but more robust, while the second incorporates more structural information but is less robust. The overall line correlation measure is defined as a linear combination of these two:

$$lncorr(i,j) = (1-w) \times lncorr_{ws}(i,j) + w \times lncorr_{cc}(i,j) \tag{12}$$

where w is a weight to be adjusted depending on the application. The two merit functions (*i.e.*, Equations 7 and 8) are then rewritten using $lncorr$ instead of $lncorr_{ws}$.

3. EXPERIMENTAL EVALUATION

In this section, we present preliminary experimental results of using our algorithm to detect tables in ASCII text and scanned image documents. The test database was composed of 25 ASCII documents (mostly e-mail messages) and 25 scanned journal pages. Each test sample was in single column format and contained one or more tables. None of the ASCII tables had ruling lines, while most scanned tables had some ruling lines. Any ruling lines detected by low level page segmentation were ignored in these experiments.

The evaluation of table detection results is itself an interesting issue for which there is currently no accepted standard metric. We present evaluation results using three different approaches in order to provide a more complete understanding of the performance.

3.1. Structural Evaluation

In considering the output from our algorithm, it becomes evident that certain classes of errors may arise as depicted in Figure 6. Relative to a ground truth, these 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.

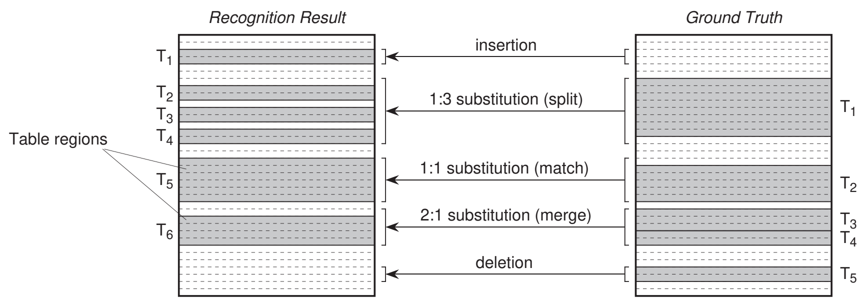


Figure 6. Various possible errors in table detection.

For the purposes of this evaluation, a table T is considered to be a region consisting of one or more contiguous text lines ranging from line i to line j inclusive, $T = [i, j]$. A document contains some number of non-overlapping (but possibly adjacent) tables listed consecutively in order of occurrence. Let $R = T_1^R T_2^R \ldots T_m^R$ be the recognized document and $G = T_1^G T_2^G \ldots T_n^G$ be the ground truth. Note that m need not equal n in general, and that T_i^R need not necessarily be the table corresponding to T_i^G for any particular i (recall Figure 6).

We define *teval* (for *table evaluation*) recursively, where $teval_{i,j}$ is the distance computed between the first i tables of R and the first j tables of G. The costs functions are: $c_{del}()$, the cost of deleting a particular table; $c_{ins}()$, the cost of inserting a particular table; and $c_{sub_{k:l}}()$, a generalized substitution cost that maps a series of k tables from one document to l tables from the other. The initial conditions are:

$$\begin{array}{lcll} teval_{0,0} & = & 0 & \\ teval_{i,0} & = & teval_{i-1,0} + c_{del}(T_i^R) & 1 \leq i \leq m \\ teval_{0,j} & = & teval_{0,j-1} + c_{ins}(T_j^G) & 1 \leq j \leq n \end{array} \tag{13}$$

and the main dynamic programming recurrence is:

$$teval_{i,j} = \min \left\{ \begin{array}{l} teval_{i-1,j} \;+\; c_{del}(T_i^R) \\ teval_{i,j-1} \;+\; c_{ins}(T_j^G) \\ \\ \min\limits_{1\leq k \leq i,\; 1 \leq l \leq j} \left[teval_{i-k,j-l} \;+\; c_{sub_{k:l}}(T_{i-k+1}^R \ldots T_i^R,\; T_{j-l+1}^G \ldots T_j^G) \right] \end{array} \right. \tag{14}$$

for $1 \leq i \leq m$, $1 \leq j \leq n$. Once the computation has completed, $teval_{m,n}$ is a measure of the similarity of the two documents in terms of their table structure. The smaller the value, the more closely the recognition result matches the ground truth. Moreover, by tracing back the sequence of optimal decisions made when evaluating the recurrence, it becomes possible recover a detailed interpretation of the errors that were determined to have arisen.

Note that the cost functions in the above formulation are completely general. For the tests that follow, we calculate a cost by aligning the tables in question on a line-by-line basis and tallying the number of lines that match. These are charged -1, while mismatches are charged 1. Although quite simple, this scheme appears to work well in practice. This value is then normalized to the target interval $[-1, 1]$, where -1 represents an exact correspondence (*i.e.*, perfect recognition relative to the ground truth) and 1 represents the opposite:

$$norm\_teval \; = \; 2\left(\frac{teval_{m,n} - min\_teval}{max\_teval - min\_teval}\right) - 1 \qquad (15)$$

where min_teval and max_teval are, respectively, the minimum and maximum possible distances for the comparison in question.

We used the above mentioned evaluation measure to study the effects of threshold on the detection of tables. This is shown for both ASCII and image documents in Figure 7. The plot shows the average over 25 documents of the structural similarity measure, $norm\_teval$, across a range of threshold values. There is a fairly large variance exhibited in the test set of 25 documents and we have examined the individual plots (they are not presented here for space reasons). This plot helps us choose a range of threshold values to obtain acceptable values of detection performance. The scales on the abscissae for the two adjacent plots are different for the ASCII and image documents. This is because the two merit functions are proportional to character and pixel spacings, respectively. The ordinates which represents the structural similarity measure, $norm\_teval$, ranges from -1 to 1 with the -1 being a perfect match between the test sample and its corresponding ground-truth and 1 being the worst possible match. Note that errors at lower threshold values (left portions of individual plots) are likely to be caused by spurious tables (insertion errors) while those at higher threshold values are more likely to be caused by missed tables (deletion errors).

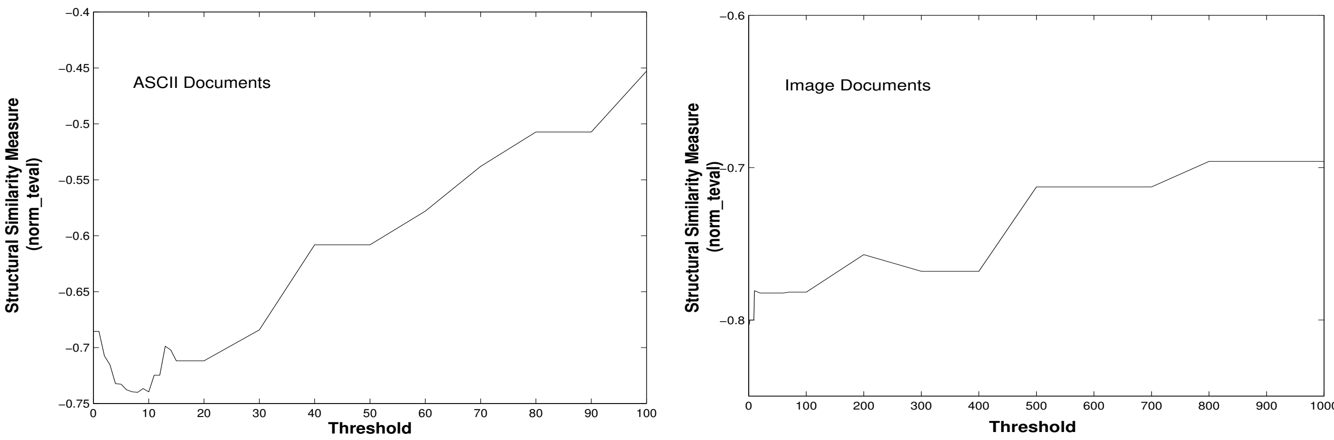


Figure 7. The effect of threshold on performance for original ASCII and scanned documents.

The effect of varying the weight in Equation 12 was also studied using this evaluation scheme. The weight w as mentioned in Equation 12 is the weight assigned to the connected component (CC) part of the table quality measure. Figure 8 shows the structural similarity measure, $norm\_teval$ of detecting tables plotted against varying values of weight, w. As before, the structural similarity measure plotted, is the average of the 25 documents in both the ASCII and the image cases. Note that in both plots, at higher values of w, the overall performance as measured by the structural similarity measure degrades. The usefulness of the CC component is more pronounced in not-so-well aligned tables and is observed in individual plots of the documents. With this plot, one can then choose the range of weights to maximize performance in detecting tables.

Finally, we used this evaluation scheme to compare the ground-truth results amongst the four authors. All possible pairwise combination of 4 individuals gives 6 such correlations. Figure 9 shows the plot of the average

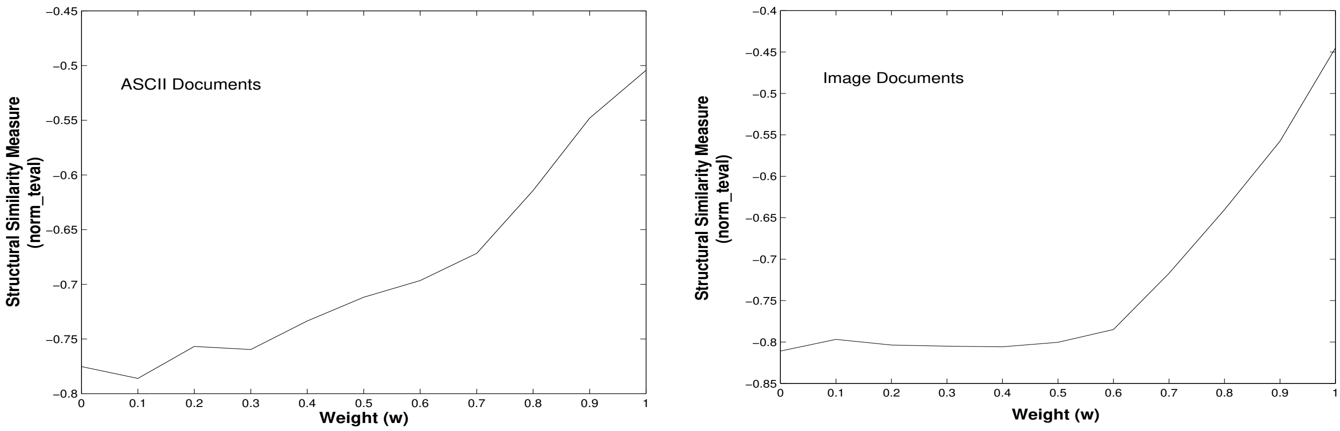


Figure 8. The effect of weight on performance for ASCII and scanned documents.

correlation for each of the 25 documents (both in the ASCII and the image case). This data shows that while there is fairly close agreement in most cases, several of the documents in our test set cause difficulties even for human interpreters (deciding what count as tables and the location of their boundaries), which demonstrates the challenging nature of the table detection problem.

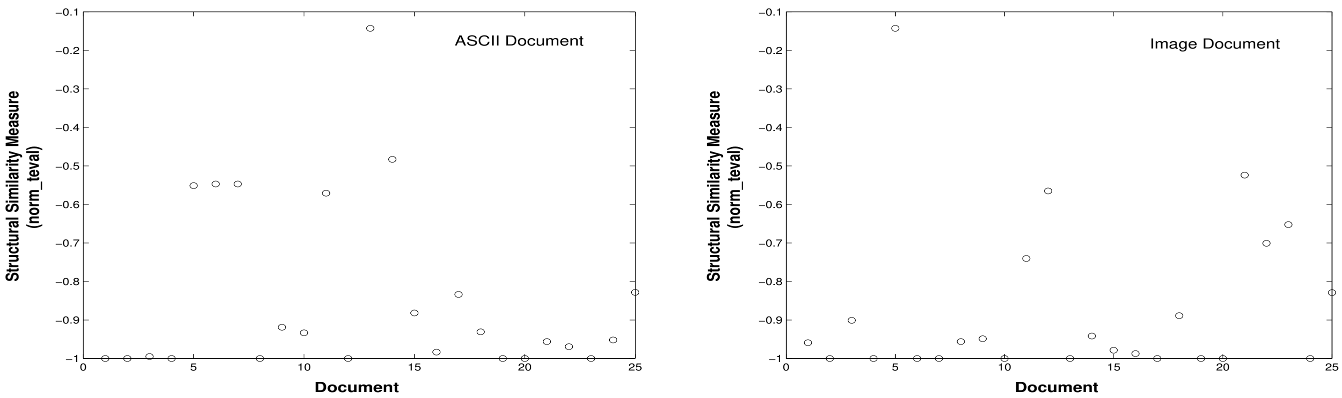


Figure 9. Ground-truth variations in ASCII and scanned documents.

One of the most common errors made by our algorithm was the splitting error. This is illustrated in Figure 10(a) where the first four lines of the original table are classified as one table and the last two lines have been classified as a second table. This is partly because of the differences in length of the first column in the table. Another common error was interpreting well-aligned text lines as tables. This is also seen in Figure 10(a) where pseudo-code is classified as a table. Merging errors tend to occur when two tables are adjacent to each other and have similar spatial structure, as shown in Figure 10(b).

3.2. Global Evaluation

Under this approach the performance of our table detection algorithm is measured in terms of the number of tables correctly identified. A table can be considered to consist of two parts: table headings and the body of the table. Due to the simplicity of our current quality measures, table headings are not always detected as part of the table. For this experiment we considered a table to be correctly detected if at least the entire body of the table was correctly detected; *i.e.*, every line in the body of the table is identified as being part of the same table, and no neighboring non-table line is included. Note that although the headers are not considered, this is still a rather strict measure. A table where all but one line in the body is detected is still considered an error. Similarly a table where all lines are detected as table lines but the table itself is split into two tables is also considered an error. Table 1 shows the precision and recall measure for both ASCII and printed documents, each at a particular threshold. In this context,

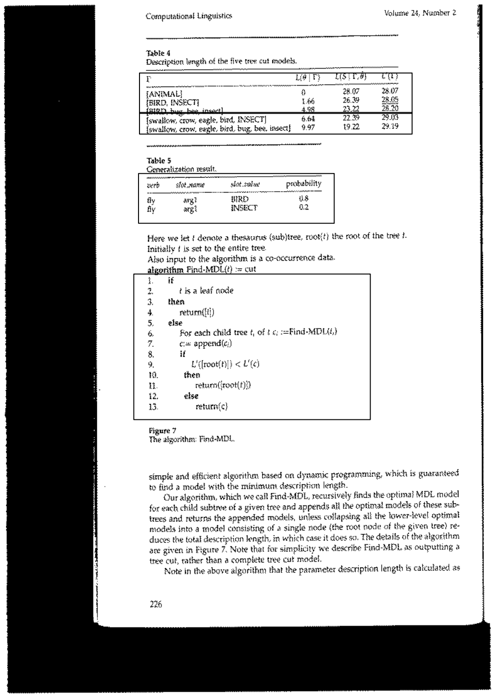

Computational Linguistics Volume 24, Number 2

Table 4
Description length of the five tree cut models.

Γ	$L(\theta \mid \Gamma)$	$L(S \mid \Gamma, \theta)$	$L'(\Gamma)$
[ANIMAL]	0	28.07	28.07
[BIRD, INSECT]	1.66	26.39	28.05
[BIRD, bug, bee, insect]	4.98	23.22	28.20
[swallow, crow, eagle, bird, INSECT]	6.64	22.39	29.03
[swallow, crow, eagle, bird, bug, bee, insect]	9.97	19.22	29.19

Table 5
Generalization result.

verb	*slot_name*	*slot_value*	probability
fly	arg1	BIRD	0.8
fly	arg1	INSECT	0.2

Here we let t denote a thesaurus (sub)tree, root(t) the root of the tree t.
Initially t is set to the entire tree.
Also input to the algorithm is a co-occurrence data.
algorithm Find-MDL(t) := cut

```
1.  if
2.      t is a leaf node
3.  then
4.      return([t])
5.  else
6.      For each child tree t_i of t c_i :=Find-MDL(t_i)
7.      c:= append(c_i)
8.      if
9.          L'([root(t)]) < L'(c)
10.     then
11.         return([root(t)])
12.     else
13.         return(c)
```

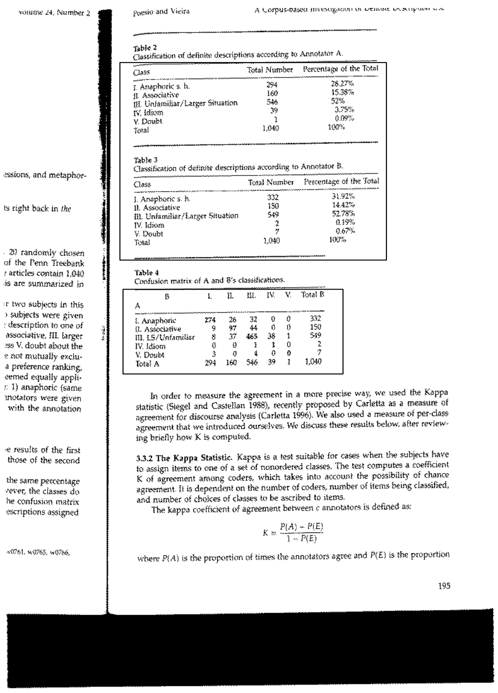
Figure 7
The algorithm: Find-MDL.

simple and efficient algorithm based on dynamic programming, which is guaranteed to find a model with the minimum description length.

Our algorithm, which we call Find-MDL, recursively finds the optimal MDL model for each child subtree of a given tree and appends all the optimal models of these subtrees and returns the appended models, unless collapsing all the lower-level optimal models into a model consisting of a single node (the root node of the given tree) reduces the total description length, in which case it does so. The details of the algorithm are given in Figure 7. Note that for simplicity we describe Find-MDL as outputting a tree cut, rather than a complete tree cut model.

Note in the above algorithm that the parameter description length is calculated as

226


Volume 24, Number 2

Poesio and Vieira A Corpus-based Investigation of Definite Description Use

Table 2
Classification of definite descriptions according to Annotator A.

Class	Total Number	Percentage of the Total
I. Anaphoric s. h.	294	28.27%
II. Associative	160	15.38%
III. Unfamiliar/Larger Situation	546	52%
IV. Idiom	39	3.75%
V. Doubt	1	0.09%
Total	1,040	100%

Table 3
Classification of definite descriptions according to Annotator B.

Class	Total Number	Percentage of the Total
I. Anaphoric s. h.	332	31.92%
II. Associative	150	14.42%
III. Unfamiliar/Larger Situation	549	52.78%
IV. Idiom	2	0.19%
V. Doubt	7	0.67%
Total	1,040	100%

Table 4
Confusion matrix of A and B's classifications.

B / A	I.	II.	III.	IV.	V.	Total B
I. Anaphoric	274	26	32	0	0	332
II. Associative	9	97	44	0	0	150
III. LS/Unfamiliar	8	37	465	38	1	549
IV. Idiom	0	0	1	1	0	2
V. Doubt	3	0	4	0	0	7
Total A	294	160	546	39	1	1,040

In order to measure the agreement in a more precise way, we used the Kappa statistic (Siegel and Castellan 1988), recently proposed by Carletta as a measure of agreement for discourse analysis (Carletta 1996). We also used a measure of per-class agreement that we introduced ourselves. We discuss these results below, after reviewing briefly how K is computed.

3.3.2 The Kappa Statistic. Kappa is a test suitable for cases when the subjects have to assign items to one of a set of nonordered classes. The test computes a coefficient K of agreement among coders, which takes into account the possibility of chance agreement. It is dependent on the number of coders, number of items being classified, and number of choices of classes to be ascribed to items.

The kappa coefficient of agreement between c annotators is defined as:

$$K = \frac{P(A) - P(E)}{1 - P(E)}$$

where $P(A)$ is the proportion of times the annotators agree and $P(E)$ is the proportion

195

(a) (b)

Figure 10. (a) Splitting (top) and insertion (bottom) errors; (b) Merging error.

precision is the percentage of detected tables that are correct and recall is the percentage of true tables that are correctly detected. Considering the strictness of this measure, the accuracies for both the image and the ASCII tables are reasonably high.

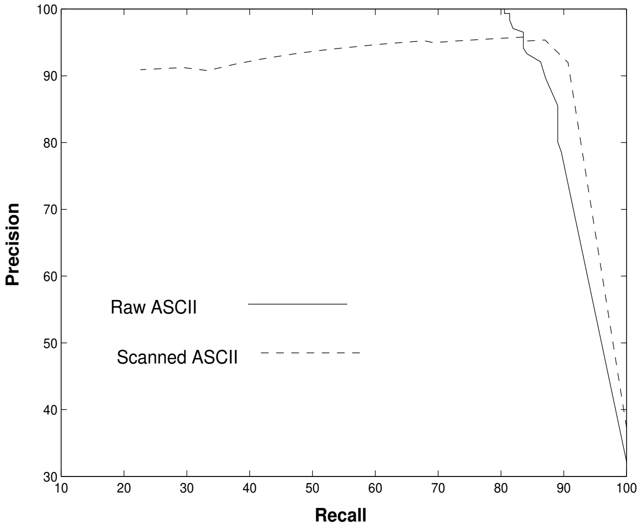
Table 1. Precision and recall for detecting ASCII and printed tables.

Medium	Recall	Precision
Image	81 %	91 %
ASCII	83 %	93 %

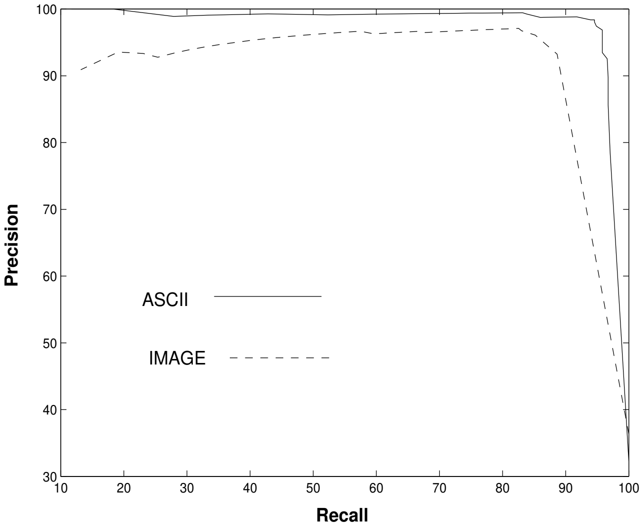
3.3. Local Evaluation

In this approach the performance of our algorithm is evaluated at the line level. After table detection, each line is labeled as either a table line or a non-table line. For a table line, no distinction is made as to which table it belongs to. In this context, precision is the percentage of detected table lines that are actually present in a true table, and recall is the percentage of true table lines identified by our algorithm. The ground-truth was generated based on the input of the four authors. Each of the authors classified every line of the input as either a table line or a non-table line. A line was classified as a table or non-table line if three or four voters classified it that way. Lines that split the vote evenly were ignored. As seen in Figure 11(a), our algorithm does well in detecting table lines in both ASCII and images.

In order to highlight the medium-independent characteristic of our algorithm, we conducted another experiment by printing the ASCII documents in 12-point Courier font and then scanning them back in. These scanned documents were then input to our table detection algorithm. The precision and recall scores for this experiment are shown in Figure 11(b). As seen from the curves, similar performance in detecting table lines was obtained with both the original ASCII and the corresponding scanned documents.



(a)



(b)

Figure 11. (a) Precision vs. recall curves for local evaluation; (b) Precision vs. recall curves for ASCII and printed/scanned versions of the same documents.

4. CONCLUSIONS

This paper describes a new approach for detecting tables across multiple media which does not rely on the presence of ruling lines or other delimiters. The approach consists of a general framework which partitions a document into a number of tables based on a set of table quality measures. The general framework is independent of the table quality measure or input format. The set of table quality measures presented has been applied to both ASCII and printed documents. Preliminary experiments on ASCII and printed documents demonstrate the effectiveness of this approach. The evaluation schemes developed in this paper are not only useful in general performance evaluation of the entire system, but also useful in improving and tuning the underlying algorithms. These evaluation schemes can be further refined based on the final application of table detection. Results from this initial analysis can be used by later stages of document processing, including table understanding and rendering. Future work includes exploring more sophisticated table quality measures such as including syntactic elements of documents and parsing the table contents.

REFERENCES

1. K. Zuyev, "Table image segmentation," in *Proc. ICDAR'97*, pp. 705–708, (Ulm, Germany), August 1997.
2. E. Green and M. Krishnamoorthy, "Recognition of tables using table grammars," in *Proc. SDAIR*, pp. 261–277, (Las Vegas, Nevada), 1995.
3. C. Peterman and C. Chang, "A system for table understanding," in *Proc. SDIUT*, pp. 55–62, (Annapolis, Maryland), 1997.
4. M. Hurst and S. Douglas, "Layout and language: Preliminary investigations in recognizing the structure of tables," in *Proc. ICDAR'97*, pp. 1043–1047, (Ulm, Germany), 1997.
5. A. Laurentini and P. Viada, "Indentifying and understanding tabular material in compound documents," in *Proc. 11th ICPR*, pp. 405–409, (The Hague, The Netherlands), 1992.
6. Y. Hirayama, "A method for table structure analysis using DP matching," in *Proc. ICDAR'95*, pp. 583–586, (Montréal, Canada), August 1995.
7. M. Rahgozar and R. Cooperman, "A graph-based table recognition system," in *Proc. of Document Recognition III, SPIE*, pp. 192–203, (San Jose, California), 1996.
8. J. H. Shamalian, H. S. Baird, and T. L. Wood, "A retargetable table reader," in *Proc. ICDAR'97*, pp. 158–163, (Ulm, Germany), August 1997.

9. T. G. Kieninger, "Table structure recognition based on robust block segmentation," in *Proc. Document Recognition V, SPIE*, vol. 3305, pp. 22–32, (San Jose, California), January 1998.
10. W. Kornfeld and J. Wattecamps, "Automatically locating, extracting and analyzing tabular data," in *Proc. SIGIR*, pp. 347–349, (Melbourne, Australia), 1998.
11. H. Baird, "Anatomy of a versatile page reader," *Proceedings of the IEEE* **80**(7), pp. 1059–1065, 1992.
12. E. Horowitz, S. Sahni, and S. Anderson-Freed, *Fundamentals of Data Structures in C*, Computer Science Press, 1993.
13. E. R. Davis, *Machine Vision, Theory Algorithms Practicalities*, Academic Press, 1997.