Table Recognition and Evaluation

Jiwon Shin
Department of Computer Science
Swarthmore College
Swarthmore, PA 19081
jiwon@cs.swarthmore.edu

Nick Guerette
Department of Computer Science
Swarthmore College
Swarthmore, PA 19081
nguerett1@cs.swarthmore.edu

Abstract

We present an algorithm that recognizes tables in document images and extracts their structural information. We use region growing to locate bounding boxes around text, and cluster them into columns by examining spatial relationships between bounding boxes and their vertical neighbors. Once initial clustering is complete, a series of post-processing steps are applied to the clusters to find columns that line up horizontally and may form tables.

1 Introduction

In performing optical character recognition on document images containing tabulated text, it is necessary to extract the text of each table cell, and desirable to obtain information about the relationships of table cells to each other.

Our goal is to create a system that recognizes tables in document images and extracts the portions of the image that correspond to each of the table cells, keeping track of the spatial relationships between table cells.

2 Previous Work

A number of researchers have suggested algorithms for table extraction and table structure recognition. A survey of the field is provided by (Zanibbi, Blostein, and Cordy , 2003).

(Watanabe et al., 1991) created a hierarchical table recognition and analysis system that first locates line segments separating table cells, uses the spatial relationships among table cells to deduce the logical relationships among them, and passes the extracted cells to higher-level processing functions. They propose allowing higher-level data processing functions to return information about contradictions encountered to lower-level functions, so that the lower-level functions can attempt a different analysis.

(Chandran and Kasturi, 1993) modified that method to require only a line at the top and bottom of a table to allow it to be recognized, but not lines separating all cells of the table. They instead use vertical and horizontal projections of binary images of extracted tables to identify boundaries between rows and between columns.

Our table recognition algorithm is based on (Kieninger, 1998). The author proposed a method that identifies tabular structures in a document by grouping word bounding boxes together and searching for vertically-aligned groups of words that could potentially be columns.

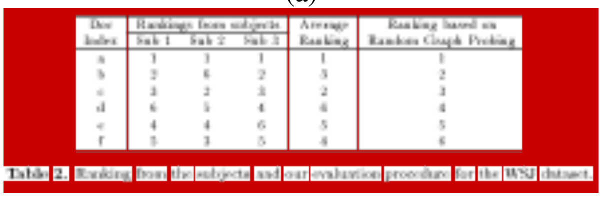
3 Algorithm

3.1 Overview

Our program takes a document image as input, and places bounding boxes around blocks of text by region growing. It then executes a set of post-processing steps to determine if any of the bounding boxes should be merged. Finally, spatial relationships between bounding boxes are examined in order to locate possible tables and identify their structure.

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 8–13
Computer Science Department, Swarthmore College

(a)



(b)

(c)

Figure 1: Segmenting by divider line identification (a) locating the divider rows (b) locating the divider columns between two divider rows (the result of the first iteration) (c) after multiple iterations; the gray regions represent areas that do not belong to a bounding box

3.2 Finding Bounding Boxes

3.2.1 "Divider" Line Method

We initially attempted to find bounding boxes by downsampling the input image and recursively locating "divider" lines. The program takes a document as input, and downsamples it to have a width in the range $[256, 512)$. It then scans across each row of the downsampled image, counting the number of times that the intensity of a pixel differs from the intensity of a neighboring pixel in that row. If the number of changes of intensity in a row is below a threshold, which is proportional to the length of the row, that row is marked as a "divider" row. A "divider" row is assumed to not go through any text (Figure 1(a)). Once all the divider rows are identified, the system executes the same procedure to scan down columns within clusters of non-dividing rows (Figure 1(b)). This procedure produces a list of bounding boxes, where each bounding box contains the minimum and the maximum row and column values. The list is added to the global bounding box queue, which is initialized to be empty. For each bonding box in the queue, we repeat the "divider" line procedure described above on the portion of image defined by the bounding box, unless the box is smaller than a threshold, and add the list of bounding boxes this procedure outputs to the queue. If the procedure returns an empty list, the bounding box is removed from the queue and is stored in a separate list of final bounding boxes. If, on the other hand, the procedure returns a list of one or more smaller bounding boxes within the original bounding box, then the original one is discarded. Bounding boxes that are smaller than a threshold are removed from the queue and added to the list of final bounding boxes. This iterative process continues until the queue is emptied.

This method of finding bounding boxes did not work well (Figure 1(c)). While the results were acceptable when there were no lines separating table cells, the algorithm's performance on tables with lines was poor. There were several problems with the algorithm. First, the downsampling process sometimes causes lines to have inconsistent intensity, resulting a failure in recognition. Second, when a table does not have many columns, the opposite can occur. Some of the rows in the table may be marked as "divider" rows because the intensity changes only few times, even though these rows are not "divider" rows. Third, when the downsampling step was skipped to attempt to correct the above problems, the algorithm generated an excessive number of bounding boxes, most of which were a character wide. This can be minimized by adding a dilation step, but because pixels that are turned "on" tend to be the same intensity, dilated columns often passed the "divider" test, failing to fundamentally fix the problem.

3.2.2 Region Growing Method

After noticing the weaknesses of the "divider" line method, we decided to implement a region growing algorithm to search for bounding boxes around words. After reading the greyscale input image, the system determines an intensity threshold using an adaptation of the ISODATA clustering algorithm (see Appendix A), and uses the threshold to binarize the image (one intensity for background, another for text). Based on the assumption that even

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 8–13
Computer Science Department, Swarthmore College

Figure 2: A set of bounding boxes that are inside a big bounding box

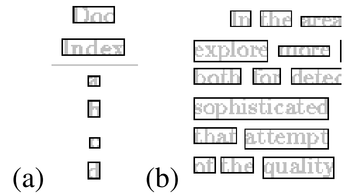


Figure 3: Two different cluster types (a) type 1 (b) type 2

a light tone of grey belongs to a letter and not the white background, we modified the ISODATA algorithm to be biased toward identifying pixels as belonging to text. Letters in the binary image are then dilated horizontally so that all characters in most words are connected. The amount by which to dilate is determined as follows. First, a histogram of lengths of horizontal runs of background pixels between text pixels is created, and the most common length is found. Then, the histogram counts of increasingly longer lengths beyond that most common length are examined, and the first whitespace length to have a count less than half that of the most common whitespace length is chosen as the amount by which to dilate. The dilation is executed by marking that many pixels to the right of each text pixel in the original image also as a text pixel.

Once the preprocessing is complete, we apply a region growing algorithm to place bounding boxes around each region of connected text pixels (with the rightmost edge of the bounding box being adjusted leftward by the amount by which text pixels were dilated earlier, so that the bounding boxes are around the original text and not the dilated text). When the image contains a table with an outline, this results in a bounding box that surrounds the table in addition to a bounding box for each word in the table. To detect this outer box, we mark all the bounding boxes whose height is greater than the average height of all the bounding boxes, and test them to determine whether or not they contain any smaller bounding boxes. We keep track of all of these “big” bounding boxes and the smaller bounding boxes they contain, as they are likely to form a table (Figure 2). This information ended up not being used in the final algorithm, however, except to ignore the “big” bounding boxes.

This method located bounding boxes more successfully and reliably, and hence, we used this method to find bounding boxes.

3.3 Table Identification

Our table identification algorithm, described below, is a modified version of the algorithm presented in (Kieninger, 1998). We had initially implemented the algorithm as it is described in the paper, which produced a set of incorrectly-grouped clusters, as expected. The problem we faced was that some of the incorrectly-grouped clusters required post-processing procedures that were too complicated, detracting from the benefits that the clustering step offered. We hence decided to modify the algorithm to do more sophisticated clustering, which simplified the post-processing step.

The table identification algorithm takes as input a list of bounding boxes that are not big bounding boxes. The first box in the list is picked as a “seed” and moved up and down by its height to test if any box in the list overlaps the region defined by the the seed box (If the height of the seed box is less than the average height of bounding boxes, we increase its height to the average height of bounding boxes for the purposes of this step). If we locate such a box, we perform the *one-to-one relation* test, which tests if the seed box has at maximum one overlapping bounding box above and one below it, and that the boxes above and below are not horizontally offset. If the found box preserves the seed box’s *one-to-one relation*, we put it into a cluster with the seed box and grow it vertically to find other boxes that need to be linked. This process continues until the *one-to-one relation* is no longer preserved or the program finds no more bounding boxes to add to the cluster. This process is then repeated on bounding boxes as yet unexamined until all bounding boxes have been clustered or found not to belong to a clus-

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 8–13
Computer Science Department, Swarthmore College

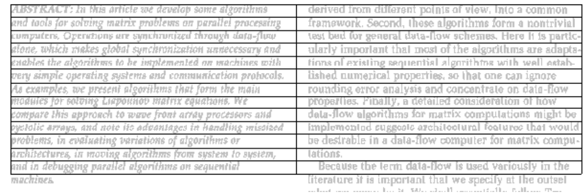
Figure 4: A false identification of a table column. The bounding boxes in gray are clustered together and are marked as *type 1* because they happen to line up vertically.

ter. The clusters generated in this process are called *type 1* clusters (Figure 3(a)), and are candidates for being columns of a table. All the bounding boxes that did not have any *one-to-one relations* with the vertical neighbors were put into a separate cluster named *type 2* (Figure 3(b)).

After we obtain clusters, we apply a series of post-processing steps to avoid errors that are created by the initial clustering process.

For each bounding box in the *type 2* cluster, we calculate the distance between it and its horizontal neighbors. A threshold for horizontal distance between neighboring bounding boxes is calculated using the same algorithm used earlier to calculate the amount by which to dilate text pixels, except examining the distance between bounding boxes rather than horizontal runs of background pixels. If the distance is less than the threshold, the seed box and the relevant neighboring box are joined into one box, and the distance between the new, larger box and its horizontal neighbors is calculated to determine whether or not more boxes should be joined to it, and this is repeated until all bounding boxes have been examined and joined if necessary. Although boxes in *type 1* clusters are not used as seed boxes in this step, to allow for large tables with closely-spaced columns, boxes in *type 1* clusters that are neighbors to a box in the *type 2* cluster being used as a seed may be joined with that seed.

After horizontal joining is completed, the same algorithm used to find *type 1* clusters is applied again on the new set of bounding boxes. The number of false identifications in blocks of text is now reduced (Figure 4). With these final *type 1* clusters, they are


ABSTRACT: In this article we develop some algorithms and tools for solving matrix problems on parallel processing computers. Operations are synchronized through data-flow alone, which makes global synchronization unnecessary and enables the algorithms to be implemented on machines with very simple operating systems and communication protocols. As examples, we present algorithms that form the main modules for solving Liapounov matrix equations. We compare this approach to wave front array processors and systolic arrays, and note its advantages in handling missized problems, in evaluating variations of algorithms or architectures, in moving algorithms from system to system, and in debugging parallel algorithms on sequential machines.

derived from different points of view, into a common framework. Second, these algorithms form a nontrivial test bed for general data-flow schemes. Here it is particularly important that most of the algorithms are adaptations of existing sequential algorithms with well established numerical properties, so that one can ignore rounding error analysis and concentrate on data-flow properties. Finally, a detailed consideration of how data-flow algorithms for matrix computations might be implemented suggests architectural features that would be desirable in a data-flow computer for matrix computations.

Because the term data-flow is used variously in the literature it is important that we specify at the outset

Figure 5: An incorrect table identification

Figure 6: An incorrect structural analysis of a table

now all compared to each other, and any clusters that contain any bounding boxes that vertically align are marked as belonging to a table.

4 Results

We tested our algorithm on twenty different document images ranging from images that only contain texts to those with pictures, figures, tables, as well as text. Our algorithm had 28.2% precision and 90.0% recall counting tables (even if rows and columns weren't correctly identified or extra rows and columns were identified outside the real table) and 40.8% precision/87.2% recall counting individual table cells identified. The most common error was an incorrect identification of a text block as a table (Figure 5). This false identification occurred especially frequently among documents that had two columns. Another error that frequently occurred was an incorrect analysis of the structure of a table. As shown in figure 6, some of the table cells are subdivided into smaller cells, causing an overproduction of table columns. The problem that occurred in figure 6 cannot be fixed easily because all the cells are in a *type 1* cluster. None of the cells will ever be tested for horizontal grouping, and the cells stay in separate columns.

Our program was able to locate tables with and without borders, tables with cells that span multiple columns (Figure 7), as well as tables of pictures (Figure 8). We had the most trouble extracting

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 8–13
Computer Science Department, Swarthmore College

			Penalty	
Subscripts		Communication	Dependence	Use
$f(i)$	$f(i)$	no	0	0
$f(i)$	$f(i)-c$	shift	c_2	c_1
$f(i)$	c	broadcast	c_4	c_3
c	$f(i)$	reduction	c_4	c_3
$f(i)$	unknown	gather	c_5	c_3
unknown	$f(i)$	scatter	c_5	c_3
$f_1(i)$	$f_2(j)$	many-to-many-multicast	c_5	c_3

(a)

Doc	Rankings from subjects			Average	Ranking based on
Index	Sub 1	Sub 2	Sub 3	Ranking	Random Graph Probing
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

(b)

Figure 7: An identification of a table with cells that span multiple columns (a) cells that span multiple columns are ignored (b) cells that span multiple columns are divided into smaller cells

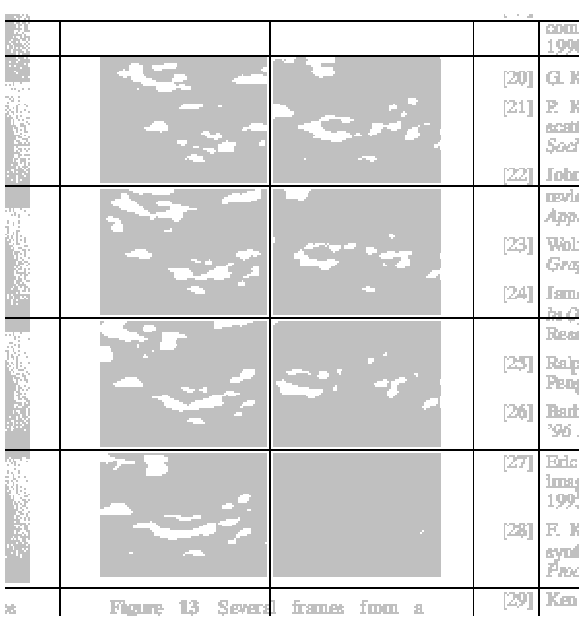


Figure 8: A correct identification of a table of pictures

the structural information of tables with cells that span multiple columns. In some cases, the cells that span multiple columns were ignored (if there were no cells in that row not spanning multiple columns) while in other cases, those cells were divided into smaller cells (if there were some cells in the same row not spanning multiple columns, so they would have been part of one of the clusters making up the table). If the latter happened, the list of table cells returned the correct information, i.e. the cell spans three columns, even though the corresponding output image did not reflect it.

5 Conclusions

We presented an algorithm that identifies tables in a document and extracts their structural information. Our algorithm finds bounding box for each unit (a word, a picture, etc.) in the document image, and clusters the bounding boxes together. The clusters go through a post-processing step, after which table cells are grouped together and tables are identified. We have shown that this algorithm works reasonably well regardless of the content of the table cells. It was capable of identifying about 90.0% of all the tables in scientific documents, whether the document was a simple document with a table on top and text in the bottom or a complex one with pictures, graphs, source codes, tables, and texts in two columns.

6 Future Work

There are several ways this algorithm can be improved. As mentioned earlier, the algorithm often returns a block of text as a table when the input document has two columns. This can be minimized by locating the column divider that divides the document into two columns, and disallowing a bounding box from one side of the divider to be tested against one from the other side.

Throughout the project, we assumed that the input image is correctly aligned. However, for documents where this algorithm is useful, such is not necessarily the case. Many documents are hand-scanned, and thus do not line up perfectly. To handle such documents, we need to add a preprocessing step to the algorithm, aligning the document before bounding boxes are located. The preprocessing step would

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 8–13
Computer Science Department, Swarthmore College

constitute determining the major and minor axes for the document and rotating the image by the discrepancy.

After the horizontal joining step, table cells spanning multiple columns are often contained in a single bounding box, and this is information that could be used, once a table is identified, to correctly assess the structure of the table by examining alignment between these bounding boxes and cells known to be in the table.

Once the algorithm extracts tables with high precision and recall, it can be integrated into an OCR software to correctly extract information in a table.

References

S. Chandran and R. Kasturi 1993. Structural Recognition of Tabulated Data *Proceedings of the Second ICDAR*, 516-519. Tsukuba Science City, Japan.

T. V. Kieninger. 1998. Table Structure Recognition Based on Robust Block Segmentation *Proceedings of Document Recognition*, volume V. 22-32. San Jose, CA.

N. B. Venkateswarlu and P. S. V. S. K. Raju 1992. Fast ISODATA Clustering Algorithms *Pattern Recognition*, volume 25(3). 335-342.

T. Watanabe, H. Naruse, Q. Luo, and N. Sugie 1991. Structure Analysis of Table-Form Documents on the Basis of the Recognition of Vertical and Horizontal Line Segments *Proceedings of the First ICDAR*, 638-646. Saint-Malo, France.

R. Zanibbi, D. Blostein, and J.R. Cordy 2003. A Survey of Table Recognition: Models, Observations, Transformations, and Inferences *http://citeseer.ist.psu.edu/zanibbi03survey.html* .

Appendix A

The following is the pseudo-code for our adaptation of the ISODATA algorithm used by the system. This version is designed to allow biasing toward either extreme. In our implementation, we used a white bias of 1 (the default value), and a black bias of 3. More information on ISODATA can be found in (Venkateswarlu and Raju, 1992).

```
thresh = range/2;

while(1) {

   black_mean = white_mean = 0;
   black_count = white_count = 0;

   for all buckets i
         in the histogram below thresh
      black_mean += i * bucket_count[i];
      black_count += bucket_count[i];

   for all buckets i
         in the histogram not below thresh
      white_mean += i * bucket_count[i];
      white_count += bucket_count[i];

   black_mean = black_mean / black_count;
   white_mean = white_mean / white_count;
   weighted_mean = ((blackmean * whitebias +
                     whitemean * blackbias)/
                    (whitebias + blackbias));

   if weighted_mean == thresh then
      break;
   else
      thresh = weighted_mean;
}

return thresh;
```


Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 8–13
Computer Science Department, Swarthmore College