An Algorithm for Identifying, Extracting and Converting Document Image Table Structures into LaTeX Format

San Sethasopon, Chidchanok Lursinsap, Peraphon Sophatsathit

Advanced Virtual and Intelligent Computing (AVIC) Center
Department of Mathematics
Faculty of Science
Chulalongkorn University
Bangkok 10330, Thailand
{Chidchanok.L[Peraphon.S]}@chula.ac.th

ABSTRACT

Paper document has been the most important and convenient source of information ever since the Chinese invented paper thousands of years ago. Paper document continues to maintain its role despite the increasing presence of the electronic counterpart. In fact, the more e-Document is deployed, the more paper document is (re)produced. This is due to volatility of electronic means and legal issues that renders hard (paper) copy the legitimate affidavit. Unfortunately, the inherent difficulty to store and retrieve such voluminous information calls for some forms of transformation from paper document to electronic, whereby facilitating subsequent automated processing. This paper focuses on such efforts in transforming one of the most common elements in paper document, i.e., tables. The proposed approach can also be applied to future research domains such as pattern recognition and computer vision.

Keywords: Table conversion, image to text, document processing, page segmentation.

1. INTRODUCTION

The advent of the Internet has brought about electronic exchange explosion. The advantages in processing speed from storage, retrieval, transmission, and dissemination make e-Document an ideal form of information representation. This does not limit merely to text-based document, but embedded image and other visual display artifacts as well, namely, tables, figures, illustrations (banners, graphs), and animation.

Versatile as it may be, e-Document suffers from several problems that are inherent in electronic nature. Storage media, archival and retrieval management systems, transmission and communications means, compatibility, and most important of all, volatility, are typical obstacles from which paper document does not suffer. The tangible value of physical evidence makes paper document one of the most important legal affidavit that its electronic counterpart can never substitute at present age. Thus, paper document still enjoys its strong hold on being the ultimate information archive.

One of the shortfalls of paper document lies in the need to convert information printed on the paper to electronic form. Numerous research endeavors have been attempted to make such needs a reality. Scanning is perhaps the most well-known and widely used approach. The scanned documents are stored as image data which poses two major disadvantages: (1) they require a large amount of storage space, and (2) it is difficult to identify and locate where the required data are stored for subsequent reuse or processing.

For efficient retrieving and storage of paper document, it is necessary to generate a description of graphical elements in the document to reduce the amount of storage space required and processing time. As a consequence, this study will focus on a popular presentation element of paper document, i.e., table, in the following domain of interest:

1. To develop an algorithm for identifying and extracting a table area from a document image.
2. To convert the table structure into LaTeX format.

The rest of the paper is organized as follows. Section 2 identifies table storage problems, applications and constraints, and potential contributions. The next two sections review some literatures relating to page segmentation and table analysis. Section 5, 6, 7, and 8 discuss the proposed algorithm for identifying, extracting, and converting a table structure from a document image into LaTeX format. Section 9 explains and compares all the experimental results. The results of this study and discussion for the future work are elucidated in the Conclusion.

2. PROBLEM IDENTIFICATION

In table areas, the columns and rows in which the characters are located are very important pieces of information. This information must be known so as to store the characters in the exact location after being extracted. To obtain this information, the table boundary must first be identified so that its structure is analyzed. The results are shape of the table and positions of the characters (being recognized by Optical Character Recognition method). This information
must be stored in some pre-arranged formats. This study will employ LaTeX format that can accommodate both table structure and the character contents as coded data. Such a scheme utilizes less space storage than other word processing formats. After all, LaTeX is one of the most widely used text formatting tools.

To obtain such precise results, a proposed algorithm was applied on gray scale document image without any skewed angles. In so doing, the algorithm can be straightforwardly integrated as a part of an Optical Character Recognition (OCR) system.

A document image can employ page segmentation method to divide the entire document into individual regions of text area and non-text area. The characters in each region of text area are processed by conventional OCR technique and stored as coded data. On the other hand, non-text regions are stored as image data. There may, however, be characters remain in these non-text areas such as the numbers in graphs, the descriptions of figures, the characters in each cell of a table, etc., which should be stored as coded data as well. Therefore, it is imperative that the proposed algorithm incorporate page segmentation and table analysis techniques so as to cope with such complications.

3. RELATED WORK IN PAGE SEGMENTATION

Earlier page segmentation methods were based on two basic approaches. The first approach is bottom-up method that starts from grouping all interested pixels together and merged them into larger homogeneous regions. The other approach is top-down method that divides the entire document image into a few large regions. These regions subsequently are recursively segmented into smaller sub-regions.

Most bottom-up methods divide text from components in a document image. L. A. Fletcher and R. Kasturi [1] described an algorithm for separating text strings from mixed texts and graphic images in 1988. This algorithm grouped each connected black pixels together to generate a connected component. It then used Hough transform [2] to group the connected components into text strings which were separated from graphic objects. This algorithm could be used with various font styles, sizes and orientations of texts.

In 1992, F. Lebourgeois, Z. Bublinski, and H. Emptoz [3] presented a bottom-up method for extracting text paragraphs and graphics from document images. The method used a horizontal smoothing from a Run Length Smoothing Algorithm (RLSA) [4] for linking characters and graphics to form blocks. The height and density of the blocks were used to identify blocks as text lines or graphics that subsequently merged the text lines into paragraphs.

Instead of considering black pixels in almost bottom-up methods, A. Antonacopoulos and R. T. Ritchings [5] considered the white pixels of background space. They presented an algorithm that linked the white pixels to build white regions in 1995. The other regions that were surrounded by the white regions were extracted into individual regions.

All of the above approaches required some suitable threshold values for linking the pixels, merging, or classifying the regions. In 1996, D. P. Mital and G. W. Leng [6] developed a technique for text segmentation that did not require any predefined parameters. This technique used two arrays to store the length of connected black and white pixels where the differences between neighbor values were considered as text regions.

Typically, a document image contains a very large amount of pixels. In order to segment a given image into blocks, it is not necessary to analyze every pixel. E. Trupin and Y. Lecourtier [7] introduced a technique that compressed the image with a compression factor $F$. Each pixel of the compressed image was set to black when there was a black pixel in an area of $F$ pixels width and $F$ pixels height. The connected black pixels of the compressed image were linked to form blocks. The application for page segmentation could be done in both bottom-up and top-down methods. A small compression factor would segment the image into words or pieces of word for processing with the bottom-up method. On the other hand, the result of a height compression factor value should process with the top-down method.

T. Saitoh and T. Pavlidis [8] presented an algorithm for page segmentation that could handle skewed pages other than rectangular columns. This algorithm would make a compressed image by considering each eight pixels width and four pixels height area. Then, each connected black pixel was linked to form blocks. Each block was classified by six classes according to its height, width, and half-tone. The connecting text blocks were merged to form one text block. After the skew angles were estimated using least square technique, some blocks were classified as tables with reference to the ruled lines. The blocks of text area were merged again into columns.

In 2001, S. Chua-aree, C. Lursinsap, P. Sophatsathit, and S. Siripant [9] presented a technique for page segmentation. This technique separated a document image to blocks of the same size. Each block was classified as text, image, and background area using Fuzzy C-mean [10] with statistical features such as mean and standard deviation. Then, the class of each block and its neighborhoods were used to define its true class.

The bottom-up method is effective for every document styles, but requires considerable computation time and memory. The top-down method, on the other hand, is faster and yields good results for the document having fixed and known structure. One of the most powerful top-down technique is RLSA, first introduced by F. M. Wahl, K. Y. Wong, and R. G. Casey [4] in 1982. The smoothing rule of RLSA would
change a white run to a black run if its length is less than or equal to a predefined threshold value, where a run is a sequence of the same value pixels. This algorithm contains the following four steps:
1. A horizontal smoothing is applied to the original document image,
2. A vertical smoothing is applied to the original document image,
3. The results of steps 1 and 2 are combined by a logical AND operation, and
4. A horizontal smoothing is applied to the output of step 3.

This four-step RLSA requires scanning the whole image four times. In 1992, it was improved to be a two-steps RLSA by F. Y. Shih, S. S. Chen, D. D. Hung, and P. A. Ng [11]. The whole image was scanned only two times. F. Y. Shih et al. used this algorithm to extract a document image into individual blocks. Each block comprised of text, horizontal or vertical lines, and graphics having properties such as the height, aspect ratio, density, and the number of changed pixels per unit length. The results of their algorithm were presented in 1996 [12].

The RLSA approach gave a good result when the input was a horizontally written document. However, it did not work well for a vertically written document. N. Amamoto, S. Torigoe, and Y. Hiroyuki [13] developed an algorithm for both horizontally and vertically written documents. The algorithm extracted a group of black pixels that were surrounded by white spaces as a block. The block, which did not have any long black runs, was decided as the horizontal or vertical writing. Then, every block was classified using similar properties to the previous approach accompanied by thin and long black lines.

In 1996, N. Papamarkos, J. Tzortzakis, and B. Gatos [14] developed a technique that calculated the threshold values for the RLSA automatically. This technique was based on the values of mean character length and mean text line distance of a document that were estimated from the distribution of the horizontal and vertical black and white runs.

Another widely used top-down method for page segmentation was Recursive X-Y Cut (RXYC) algorithm. The RXYC cut an image recursively into blocks. At each step of the recursive process, the algorithm computed the sum of black pixels along horizontal and vertical axes to create document projection profile. This document projection profile was a waveform whose deep valleys corresponded to blank areas of the document. A deep valley with a width greater than a threshold could be cut as the edge of a block. H. Wang, S. Z. Li, and S. Ragupathi [15] used this algorithm for document segmentation and explained a classification process with tested results in 1997.

Page segmentation by RLSA or RXYC uses a fixed threshold for the entire document, so there might be problems with some document styles such as a text lines with different font size or font styles, etc. To solve these problems, K. C. Chan, X. Huang, and P. Bao [16] presented a page segmentation method based on the concepts of fuzzy set theory. This method determined the thresholds in any positions in the document automatically.

Both bottom-up and top-down methods have their own advantages. As such, attempts were made to combine these two methods. Y. Hirayama [17] presented a block segmentation method that started in bottom-up fashion by linking black connecting pixels to form rectangles. They were classified as character strings, horizontal and vertical lines, and picture elements based on their properties. These character strings were merged into text groups. The threshold of merging was determined by analyzing the height and distance of character strings. In the meantime, top-down approach started detecting the borderlines of columns by linking the edges of text groups. As a consequence, the entire page was segmented into blocks of text and picture areas based on the borderlines and projection profile.

In 1998, A. K. Jain and B. Yu [18] presented an algorithm for page segmentation based on a top-down method which was constructed from bottom up. They defined blocks of regular small connected components as text regions and large connected components as non-text regions.

4. RELATED WORK IN TABLE ANALYSIS

Table analysis involves table identification and table recognition. The goal of table identification is to separate table areas from non-table areas in a document image. Each table area contains important data that are related by positions (column and row). Table recognition determines the structure of the table area and stores the table in some pre-arranged formats.

Typical page segmentation methods encompass table identification capability to identify embedded table areas by considering blocks that have different characteristics from conventional text blocks during page segmentation process. For example, T. Saitoh and T. Pavlidis [8] classified some blocks as table area based on ruled lines. They defined a block as a table area if it had at least two horizontal lines (at top and bottom sides) and one vertical line (not so close to the side edges).

N. Amamoto et al. [13] defined a block as a table area according to certain properties of the block. A table block had to have a number of thin and long solid black lines higher than other classes.

A. K. Jain and B. Yu [18] identified table areas from non-text blocks by first extracting horizontal and vertical lines. The horizontal top and bottom lines should have the same length without any skew lines. The height of all connected components should be small.

Y. Hirayama [19] presented a process for table identification from non-text blocks that had both horizontal and vertical lines. The block was assumed to be
a table area and was further divided into small areas by the horizontal and vertical lines. Each small area was classified as a celled area or non-celled area. If the non-celled area had less space than one thirds of the whole block area, the block should be a table area.

Y. Wang, I. T. Phillips, and R. Haralick [20] presented a table identification algorithm that analyzed background. They used large horizontal blank blocks to construct table candidates. The table candidates were then identified based on some predefined parameters such as the ratio of total large vertical blank block area over the table candidate area, the maximum difference of the cell baselines on a row, and the maximum difference of the justification in a column. These parameters could be estimated from any real table instances.

Most approaches did not concentrate on table identification but focused on table recognition. S. Chandran and R. Kasturi [21] used the horizontal and vertical projection profile of a table image to extract data as cells in the table. Each cell was labeled by a number and arranged in order of positions (left to right and top to bottom) as shown in Figure 1. The data and its corresponding label were stored with the number of columns in the table.

However, the above method could not be used for complicated tables. K. Itonori [22] presented a method that was applicable for all styles of tables. It used the projection profile to assign row and column numbers in a table image. Each cell could be stored with its right and bottom coordinates as shown in Figure 2.

E. Green and M. Krishnamoorthy [23] gave each cell a label by a series of letters for columns and numbers for rows. The label consisting of one letter followed by a number was used for each level of cell in the table image. An example of a complete labeling appears in Figure 3.

J. F. Arias, A. Chhabra, and V. Misra [24] used the information about horizontal and vertical lines to extract the structure of a table image. They used bit strings for labeling each cell of the table. The label indicated the column and row to which the cell belonged. The length of the label was equal to the number of columns and rows that contained in the table. Each bit indicated an order of column and row. The bit was 1 if the cell corresponded to a given column and row. An example of bitstring cell labeling of a table is shown in Figure 4.

T. Watanabe, Q. Luo, and N. Sugie [25] used a binary tree to represent the structure of a table. Each node corresponded to a block of one or more cells. The nodes were arranged in vertical node \( v \), horizontal node \( h \), and the terminal node \( t \). The type of node represented the relation to its left and right child nodes as illustrated in Figure 5. Figure 6 presents a sample of table structure and its corresponding binary tree.

H. Saiga, Y. Kitamura, and S. Ida [26] extracted all dotted and solid lines in a table image and defined their names. Then, they constructed cells from the horizontal and vertical line intersections. Each cell was stored by name which referred to the horizontal and vertical lines that formed it. This algorithm was easily combined with character recognition.

Y. Hirayama [19] presented a dynamic programming matching method to detect the correspondence between strings in two columns. They applied this method to multi-column tables. As a result, the information arrangement was easily detected and transformed into other data formats.
5. MODEL AND ALGORITHM

This Section explains the dichotomy of the proposed model of this study to be processed in three steps. Firstly, a document image is extracted from individual blocks. Next, each block is identified as a table area or a non-table area. Lastly, the table area is converted into LaTeX format.

The document image used in this study is a gray scale image. The value of each pixel ranges from 0 to 255 to denote color intensity. The pixels whose value is close to 0 are called black pixels. On the contrary, the value of white pixels is close to 255. A table image from the scanned document is stored in a two dimensional array, whereby the number of columns and rows define the width and height of a table, respectively.

5.1 BLOCK EXTRACTION APPROACH

This study focuses on the table areas of a document image. All table areas have rectangular shapes. Thus, the proposed algorithm will extract a document image to rectangular blocks that are easy for subsequent processing. In block extraction, an image smoothing algorithm is applied to a document image to combine neighboring pixels of the block. The result of image smoothing is a blurred document image. Each pixel of the blurred image is determined by the adjacent pixels according to the following equation:

\[ B_{i,j} = \frac{\sum_{i=-I}^{I} \sum_{j=-I}^{I} P_{k,l}}{(2I+1)^2} \]  \hspace{1cm} (1)

where

- \( B_{i,j} \) is the value of the pixel in the \( i^{th} \) row and \( j^{th} \) column of the blurred image.
- \( P_{k,l} \) is the value of the pixel in the \( k^{th} \) row and \( l^{th} \) column of the document image.
- \( I \in \{1, 2, 3, \ldots, N\} | N \) is the number of columns.

The value of \( I \) depends on the space between text lines in the document image.
Fig. 7: An example of process in the block extraction method: (a) a document image, (b) a blurred image, (c) a blurred image with the cutting lines, (d) the result of applying the cutting lines on the document image.

We consider the blurred image to construct cutting lines that are used for extracting the document image to rectangular blocks. The blurred image is scanned horizontally or vertically from one edge to the opposite edge. The scanned results are classified into two types. The first type has only white pixels that are designated as blank lines. The second type has some black pixels which become data lines. A horizontal cutting line is a data line that is next to the blank lines. An algorithm for constructing the horizontal cutting line is as follows:

Cutting Lines Construction Algorithm
1. If the first row is a data line then
2. Set this row as the top edge of a new block.
3. For i=2 to number of row do
4. If this row is a data line
5. AND the (i - 1)th row is a blank line then
6. Set this row as the top edge of a new block.
7. If this row is a blank line
8. AND the (i - 1)th row is a data line then
9. Set the (i - 1)th row as the bottom edge of a block.
10. End for
11. If the last row is a data line then
12. Set the last row as the bottom edge of a block.

The algorithm is repeated three times. The first time is mainly for constructing the horizontal cutting lines that extract the blurred image into smaller blocks. The second time is applied on each column in each block for vertical cutting lines. The third time is for the horizontal cutting lines that lie inside the vertical block. All cutting lines are used for extracting the document image into rectangular blocks. Figure 7 illustrates an example of the process in the blocks extraction method.

5.2 TABLE IDENTIFICATION METHOD
Each block from the last step is assumed as a table block. The groups of black pixels in every block represent lines and data of the table. Data and lines are then analyzed and stored in a text file that is suitable for subsequent identification and conversion to LaTeX format. The text file preparation process starts by determining the positions of the lines enclosing each block. This block is then split into small rectangular areas using the lines as their border. The coordinates of each area are calculated from the information of the enclosing lines, namely, top, left, bottom, and right positions. The algorithms for line detection and block separation are given below.

Lines Detection Algorithm
1. List L is empty.
2. For each row of a block do
3. If there are some black runs that are longer than α then
4. Store their beginning and ending positions as the row number in list L.
5. End for
6. For each column of a block do
7. If there are some black runs that are longer than one fifth of the number of rows then
8. Store their beginning and ending positions as the column number in list L.
9. End for

Block Separation Algorithm
1. List B is empty.
2. For i = 1 + β to number of rows - β do
3. For j = 1 + β to number of columns - β do
4. If Wij is a white pixel and not in the range of the coordinates in list B then
5. Set Wij to be the point of a new borderline.
7. If there are some white pixels around Wij then
8. Do
9. Find the next pixel of this borderline.
10. Start at the opposite of the last direction as illustrated in Figure 8.
11. Find a white pixel around the point in clockwise direction.
12. Stop when a white pixel is found.
13. Set this pixel to be the new point of consideration.
14. Until the point is Wij
15. Define the coordinate of this rectangular area from the pixel positions in this borderline.
16. Add the coordinates of the area to list B.
17. End for
18. End for
19. For all rectangular areas in the list do
20. For each edge of the area do
21. If the edge is close to the border of table block and there is no line from list L lying between the border and the edge then
22. Define this edge to contain no line.
23. Else
24. Define this edge to contain a line.
25. End for
26. End for
27. End for
Fig. 8: Search directions starting from the middle pixel.

Data Block Coordinate Defining Algorithm

1. List $D$ is empty.
2. For each rectangular area in list $B$ do
3. For $i = 1$ to the last row of this area do
4. For $j = 1$ to the last column of this area do
5. If $B_{i,j}$ is a black pixel and not in the range of the coordinates in list $D$ then
6. Set $B_{i,j}$ to be the point of a new borderline.
7. Direction = 1.
8. If there are some black pixels around $B_{i,j}$ then
9. Do
10. Find the next pixel of this borderline.
10.1 Start at the opposite of the last direction as illustrated in Figure 8.
10.2 Find a black pixel around the point in clockwise direction.
10.3 Stop when a black pixel is found.
11. Set this pixel to be the new point of consideration.
12. Until the point is $B_{i,j}$
13. Define the coordinate of this data block from the pixel positions in this borderline.
14. Add the coordinates of the data block to list $D$.
15. End for
16. End for
17. For all data blocks in the rectangular area do
18. If the size is too small then
19. Remove it from list $D$.
20. End for
21. If there are more than one data block in the rectangular area then
22. For all data blocks in the rectangular area do
23. For each edge of the data block do
24. If the edge position is close to the border of rectangular area then
25. Set the border as the edge.
26. Else
27. Define this edge to contain no line.
28. End for
29. End for
30. End for
31. End for
32. End for
33. While some edges are not equal to their boundary do
34. For each data block do
35. Decrease the value of its top edge.
36. Increase the value of its left edge.
37. Increase the value of its bottom edge.
38. Increase the value of its right edge.
39. End for
40. End do
41. For each area that is not covered by any table cells do
42. If the edge of the cell that connects to the area does not have the line then
43. Add this area to that cell.
44. End for

Every data block expands to form a rectangular cell that fits in the table. The boundary of the data block is designated by the adjacent data blocks in four directions (top, left, bottom, and right). We then find the relation of each data block to set the row or column. Data blocks with the same value of the left, right, or center position are assigned to the same column. Hence, the data blocks in the same column must have the same left and right boundary values. Similarly, the data blocks in the same row must have the same top and bottom boundary values. There may be some areas in the block which are not covered by any table cells. In which case, we add those areas to the adjacent cells, provided that the edges of the adjacent cells do not have lines separating them. Figure 9 shows an example of table cell formation.

Table Cell Forming Algorithm

1. For each data block do
2. If there are some data blocks over it then
3. Set its top boundary = the bottom edge of the nearest data block on the top.
4. Else
5. Set its top boundary = 1.
6. If there are some data blocks under it then
7. Set its bottom boundary = the top edge of the nearest data block on the bottom.
8. Else
9. Set its bottom boundary = the number of pixel height.
10. If there are some data blocks to the left of it then
11. Set its left boundary = the right edge of the nearest data block on the left.
12. Else
13. Set its left boundary = 1.
14. If there are some data blocks to the right of it then
15. Set its right boundary = the left edge of the nearest data block on the right.
16. Else
17. Set its right boundary = the number of pixel width.
18. End for
19. Construct the rows of the table.
20. Construct the columns of the table.
21. For each row of the table do
22. For each data block in this row do
23. Set its top boundary = max [all top boundaries in this column].
24. Set its bottom boundary = min [all bottom boundaries in this column].
25. End for
26. End for
27. For each column of the table do
28. For each data block in this column do
29. Set its left boundary = max [all left boundaries in this column].
30. Set its right boundary = min [all right boundaries in this column].
31. End for
32. End for
33. While some edges are not equal to their boundary do
34. For each data block do
35. Decrease the value of its top edge.
36. Increase the value of its left edge.
37. Increase the value of its bottom edge.
38. Increase the value of its right edge.
39. End for
40. End do
41. For each area that is not covered by any table cells do
42. If the edge of the cell that connects to the area does not have the line then
43. Add this area to that cell.
44. End for

After table cell formation is complete, the cell coordinates, lines, and data information are stored as a text file. Figure 10 shows an example of this text file. The first row denotes the number of cells and the size of the block. The remaining rows store the line coordinates and data information for each cell. Each column is denoted by a cell number, top position, left
Fig. 9: An example of table cell formation: (a) a sample block, (b) the rectangular areas, (c) the coordinate boxes of the data blocks, (d) the coordinate boxes of the table cells.

Fig. 10: A text file example of cell coordinates and line information.

position, bottom position, right position, line information of top edge, left edge, bottom edge, right edge, and the appearance of the data inside the table cells.

This text file is used for identifying the table blocks based on the arrangement of the cell and line positions. Every block is initially assumed to be a table block. Two conditions are utilized to validate any non-table blocks. The first condition imposes that any two cells should not overlap each other as depicted in Figure 11. The second condition checks the appearance of the border lines. Cell $C_i$ can overlap cell $C_j$ in four scenarios:

1. The top position of $C_i$ is between the top and bottom position of $C_j$, while the bottom position of $C_i$ is greater than the bottom position of $C_j$.
2. The bottom position of $C_i$ is between the top and bottom position of $C_j$, while the top position of $C_i$ is less than the top position of $C_j$.
3. The left position of $C_i$ is between the left and right position of $C_j$, while the right position of $C_i$ is greater than the right position of $C_j$.
4. The right position of $C_i$ is between the left and right position of $C_j$, while the left position of $C_i$ is less than the left position of $C_j$.

Fig. 11: Illustration of the possibilities of cell overlapping.

After cell overlap check is complete, the second condition is applied to separate a matrix block from a table block. A matrix has two vertical lines, one on the left and the other on the right side of the block. The cells are uniform and arranged consecutively. There may not be any overlapping cells in a matrix. Consequently, a table block identification algorithm is needed to distinguish a table from a matrix as the text file is processed.

Table Identification Algorithm

1. Let $T = 1$.
2. For each table cell do
3. If it overlaps other table cells then
4. $T = 0$.
5. End for
6. If $T = 0$ then
7. This block is not a table block.
8. Else
9. If there are only two lines on the left and right sides then
10. This block is not a table block but a matrix.
11. Else
12. This block is a table block.

5.3 CONVERGING TABLE INTO LaTeX FORMAT

The table block, which is identified in the previous step, will be further processed so that structural information from the text file gets converted into LaTeX format. LaTeX can align text in columns as a table by means of tabular environment. From left and right positions in the text file, the required number of columns can be determined at the beginning of the tabular environment. We start a LaTeX code with \begin{tabular}{cccc}, followed by the command for the line on the top of the table. The number of c in cccc is equal to the number of columns. The command for the line takes on two forms, namely, \hline and \cline. The \hline command is used if there is a line along the row. If the lines lie in some cells on the same row, the \cline command is used instead. Next, we determine the number of cells in each row to specify the number of columns and positions of the line. For each cell, we use the
Fig. 12: An example of a table conversion LaTeX file.

```
<table>
<thead>
<tr>
<th>TEXT</th>
<th>TEXT</th>
<th>TEXT</th>
<th>TEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXT</td>
<td>TEXT</td>
<td>TEXT</td>
<td>TEXT</td>
</tr>
<tr>
<td>TEXT</td>
<td>TEXT</td>
<td>TEXT</td>
<td>TEXT</td>
</tr>
<tr>
<td>TEXT</td>
<td>TEXT</td>
<td>TEXT</td>
<td>TEXT</td>
</tr>
<tr>
<td>TEXT</td>
<td>TEXT</td>
<td>TEXT</td>
<td>TEXT</td>
</tr>
<tr>
<td>TEXT</td>
<td>TEXT</td>
<td>TEXT</td>
<td>TEXT</td>
</tr>
<tr>
<td>TEXT</td>
<td>TEXT</td>
<td>TEXT</td>
<td>TEXT</td>
</tr>
</tbody>
</table>
```

Fig. 13: An example of a table generated from converted LaTeX file.

```
\begin{multicolumn}{l}{l}{\text{\textbackslash multicol}}\text{\textbackslash column}\text{\textbackslash n} \text{\textbackslash \textbackslash | command to denote the cell width.}
```

A “|” is put between two cells if there is a vertical line in that position. An “\” is used to separate each cell. Each row is delimited by the “\}” command. Figure 12 and Figure 13 show examples of a LaTeX file and a table generated from this algorithm.

6. EXPERIMENTAL RESULTS

The proposed algorithm is implemented by C programming language. The tested samples consist of actual blocks extracted from real document images and synthesized blocks. Seven document images having small skewed angles and noises are extracted and placed in rectangular blocks. The contents of these blocks consist of text paragraphs, pictures, graphs, tables, and matrices that are identified as table blocks and non-table blocks.

The table blocks with various styles of lines and data arrangement are synthesized for testing. Table 1 presents the tested samples that are correctly identified. Test results of samples table area and non-table area are summarized in Table 2 and Table 3. Table 4 depicts the results of identification method. Most table blocks are correctly identified and converted. Comparison of the table blocks and their respective structure obtained from LaTeX are shown in Table 5. Nevertheless, some mis-identifications were resulted due to excessive cell overlapping. Figure 14 shows an example of a table misidentified as a non-table block because of cell overlapping in the circles.

7. CONCLUSION

A new algorithm is proposed in order to identify various styles of table from scanned documents. The extracted tables are converted into LaTeX that is suitable for modification, storage, retrieval, and transmission. The proposed algorithm examines the line appearance and the position of cells inside the table in determining the table area. Thus, a matrix will be identified as a non-table area. In the case of a table without any lines, the algorithm uses only the position of cells as the parameter for decision.

From the experimental results, this algorithm can correctly identify many types of table. However, the algorithm fails to distinguish the table if there exists pockets of excessive overlapping as appeared in Figure 4 and Figure 14.

Experimental results show that the algorithm yields a 94.12% correct identification rate for the actual table area and a 100% for the synthesized ones. Future enhancements can be considered in the following issues:

1. Develop a faster method for preparing cell and line positions.
2. Support tables that encompass non-table areas.
3. Establish detailed information of block classification for more sophisticated LaTeX representations.

8. ACKNOWLEDGMENTS

A special thank to Miss Atchara Mahaveerawat for her tireless assistance to fix some of the manuscript typesetting problems.
### Table 1

<table>
<thead>
<tr>
<th>Type of Cell</th>
<th>Matrix</th>
<th>Ash</th>
<th>Carbon</th>
<th>Nitrogen</th>
<th>Number of cells</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type A</td>
<td>524</td>
<td>24.4</td>
<td>17.6</td>
<td>2.8</td>
<td>50</td>
</tr>
<tr>
<td>Type B</td>
<td>772.7</td>
<td>18.4</td>
<td>16.2</td>
<td>2.9</td>
<td>50</td>
</tr>
<tr>
<td>Type C</td>
<td>78.3</td>
<td>16.7</td>
<td>10.5</td>
<td>2.4</td>
<td>50</td>
</tr>
<tr>
<td>Type D</td>
<td>9.1</td>
<td>1.2</td>
<td>10.8</td>
<td>2.3</td>
<td>50</td>
</tr>
</tbody>
</table>

### Table 2

<table>
<thead>
<tr>
<th>Country</th>
<th>Number of deaths</th>
<th>Number of deaths measured</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brazil</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>Canada</td>
<td>150</td>
<td>150</td>
</tr>
<tr>
<td>China</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>Hong Kong</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Japan</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>Sweden</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

### Non-table

- Connectivity (13) similarity (14) and noise connectivity. Especially, in spite of boundary noise, our method, rectangle detection is the 'x' shapen or 'T' shaped rectangles. Table 3 summarizes the appropriate sharing algorithms to each map.

### Non-table

- \[
  \sqrt{\text{Min}_{i,j}^2} \leq \left[ \sqrt{\text{Min}_{i,j}^2} \right] + \left[ \sqrt{\text{Max}_{i,j}^2} \right] 
  \] (50)

### Non-table

- Higher bars indicate better performance.

### Non-table

- Fig. 4 Degraded image.
Table 2: The table samples.

<table>
<thead>
<tr>
<th>Table</th>
<th>Components</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Line</td>
</tr>
<tr>
<td>Actual</td>
<td>-</td>
</tr>
<tr>
<td>Synthesis</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3: The non-table samples.

<table>
<thead>
<tr>
<th>Non-table</th>
<th>Picture</th>
<th>Graph</th>
<th>Diagram</th>
<th>Matrix</th>
<th>Others</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>-</td>
<td>20</td>
</tr>
<tr>
<td>Synthesis</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 4: Tested results of table identification.

<table>
<thead>
<tr>
<th>Samples</th>
<th>Number of tests</th>
<th>Results</th>
<th>Correct</th>
<th>Miss</th>
<th>Correct identification rate (%)*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table</td>
<td>Actual</td>
<td>17</td>
<td>16</td>
<td>1</td>
<td>94.12</td>
</tr>
<tr>
<td></td>
<td>Synthesis</td>
<td>19</td>
<td>19</td>
<td>-</td>
<td>100</td>
</tr>
<tr>
<td>Non-table</td>
<td>Actual</td>
<td>28</td>
<td>28</td>
<td>-</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Synthesis</td>
<td>5</td>
<td>5</td>
<td>-</td>
<td>100</td>
</tr>
</tbody>
</table>

*Correct identification rate = Number of correct results/Number of tests
### Table 5: Comparison of the table blocks and their structures obtained from LaTeX.

<table>
<thead>
<tr>
<th>Table</th>
<th>Block 1</th>
<th>Block 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TEXT</td>
<td>TEXT</td>
</tr>
<tr>
<td></td>
<td>TEXT</td>
<td>TEXT</td>
</tr>
<tr>
<td></td>
<td>TEXT</td>
<td>TEXT</td>
</tr>
<tr>
<td></td>
<td>TEXT</td>
<td>TEXT</td>
</tr>
</tbody>
</table>

---

### Table 3. Performance of thinning algorithms for water and sewer map.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time (s)</th>
<th>Connectivity</th>
<th>Number of noisy branches</th>
<th>Branch similarity</th>
<th>Erosion of end point (pixel)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPTA</td>
<td>31.5</td>
<td>perfect 8</td>
<td>75</td>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td>CGT</td>
<td>14.6</td>
<td>perfect 8</td>
<td>187</td>
<td>3</td>
<td>14</td>
</tr>
<tr>
<td>Areal(L=5)</td>
<td>26</td>
<td>perfect 8</td>
<td>10</td>
<td>6 (poor)</td>
<td>49</td>
</tr>
<tr>
<td>Chen</td>
<td>19.3</td>
<td>perfect 8</td>
<td>10</td>
<td>4</td>
<td>25</td>
</tr>
<tr>
<td>Wang</td>
<td>36</td>
<td>imperfect 8</td>
<td>9</td>
<td>5</td>
<td>14</td>
</tr>
<tr>
<td>Holt</td>
<td>19</td>
<td>perfect 8</td>
<td>13</td>
<td>1</td>
<td>15</td>
</tr>
</tbody>
</table>

---

### Table: Blade Center and Blade Center H20

<table>
<thead>
<tr>
<th>Feature</th>
<th>Blade Center</th>
<th>Blade Center H20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blade server type</td>
<td>128</td>
<td>Intel Xeon processor</td>
</tr>
<tr>
<td>Standard model</td>
<td>CPU+ROM</td>
<td>Number of processors (integrated)</td>
</tr>
<tr>
<td>Cooling module</td>
<td>2 hot swap</td>
<td>Memory: Up to 8GB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Network: Dual integrated</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Operating system: Microsoft Windows 2000</td>
</tr>
</tbody>
</table>

---

![Image 1](image1.png)

![Image 2](image2.png)
References


Name: Chichanok Lurinsap

Affiliation: Advanced Virtual and Intelligent Computing Center (AVIC), Department of Mathematics, Faculty of Science, Chulalongkorn University

Address: Phyahtai Road, Patumwan, Bangkok, 10330 Thailand

Brief Biographical History:
1978-1979 Department of Computer Engineering, Chulalongkorn University, Thailand
1982-1986 Department of Computer Science, University of Illinois at Urbana-Champaign, USA
1986-1995 Center for Advanced Computer Studies, University of Louisiana, USA
1995-present Department of Mathematics, Chulalongkorn University, Thailand

Main Works:

Membership in Learned Societies:
- IEEE
- Neural Network Society
- Sigma Xi
- Royal Institute of Thailand
Name: Peraphon Sophatsathit

Affiliation: Advanced Virtual and Intelligent Computing Center (AVIC), Department of Mathematics, Faculty of Science, Chulalongkorn University

Address: PhyaTHAI Road, Patumwan, Bangkok, 10330 Thailand.

Brief Biographical History:
1994-1997 National Electronics and Computer Technology Center (NECTEC), Bangkok, Thailand
1997-present Department of Mathematics, Faculty of Science, Chulalongkorn University

Main Works:

Membership in Learned Societies:
• IEEE