A Fast Preprocessing Method for Table Boundary Detection: Narrowing Down the Sparse Lines using Solely Coordinate Information

Ying Liu, Prasenjit Mitra, C. Lee Giles
College of Information Sciences and Technology
The Pennsylvania State University
University Park, PA, 16802
{yliu, pmitra, giles}@ist.psu.edu

Abstract

As the rapid growth of PDF document in digital libraries, recognizing the document structure and detecting specific document components are useful for document storage, classification and retrieval. Tables, as a specific document component, are ubiquitous everywhere. Accurately detecting the table boundary plays a crucial role for the later table structure decomposition and table data collection. In this paper, we propose an easy but effective table boundary detection method. Our method has two unique advantages comparing with other works in this field: 1) Because most tables are text-based, we claim that the text object of PDF itself is good enough for table detection. In addition, we believe that the font information is not so reliable as other works stated. 2) Based on the nature of the table cells, we notice the sparse-line *property of table rows. By filtering out the non-sparse lines initially, the table boundary detection problem can be simplified into the sparse line analysis problem easily. The experimental results not only confirm the importance of the coordinate information, but also demonstrate the effectiveness of sparse lines in the table boundary detection. Combining with other keywords, our method is even applicable to detect other document components (e.g., mathematical formula or the references).*

1 Introduction

Portable document format PDF is a widely used document format in digital libraries because it can preserve the appearance of the original document. Although a good number of research has been done to discover the document layout by converting the PDFs to other types of files (e.g., image, html, text) in the past two decades, automatically identifying the document logical structures information (e.g., words, text lines, paragraphs, etc) and extracting the document components (e.g., figures, tables, mathematical formulas, etc) as well as the content [2] are still a challenging problem. The main reasons includes: 1) the structural information is not explicitly marked up because of the untagged nature of PDF format; 2) the text sequences are often messily generated by the existing text extraction tools; 3) some new noises are incurred by some necessary tools (e.g., OCR), if converting the PDFs into other media (e.g., image).

Table, as a specific document component, is ubiquitous everywhere. For the further table content analysis, locating the boundary of a table in a document is the first and a crucial step (e.g., the table data extraction and the table search). By observing diverse tables, we notice that all the table boundaries share an important feature: almost all the lines belonging to the table areas are sparse in the perspective of the text density. In this paper, we propose a novel but effective method to quickly locate the boundary of a table in a document by taking advantage of this property. In comparison with other existing approaches that take much effort to study most of the document information, our approach solely uses the text objects in PDF instead of analyzing all the objects as other researchers advocated. In stead of considering other text style information, we only adopt the coordinate information in the text object to detect the table boundaries by analyzing the labeled sparse lines. This method is also applicable to the boundary detection of other document components that share the same sparse-line property, such as the formula and the reference.

Our approach has the following advantages: 1) We only need to check the text objects without having to worry about the object segmentation performance in PDF documents; 2) Text is easy to obtained, with many text extractor tools (e.g., Xpdf[1], PDF2Text[2], PDFBOX[3], The PDFlib Text Extraction

[1]http://www.foolabs.com/xpdf
[2]http://www.pdf2text.com/
[3]http://www.pdfbox.org/

Toolkit (TET)[4], PDFTextStream[5], etc); 3) We can easily get rid of the non-sparse lines from a PDF document page, which saves much effort for the further works; Moreover, the non-sparse lines usually incur errors at the document resorting work and the later table structure decomposition phase; 4) Because almost all the table body lines are covered by the sparse lines, we can easily get more details about the tables by zooming in the sparse lines. Keywords can be adopted to facilitate the identification of the table starting and ending places;5) This method is applicable to other document components, such as the mathematical formula and reference; 6)The performance is competitive comparing with the works considering all the objects in PDF documents and other factors such as the font size;

The rest of the paper is organized as follows. Section 2 is the related work. Section 2 introduces the sparse-line property of the table lines based on the observations. Section 3 explains why the text object listed by the PDF document content stream and the coordinate information provide enough information for the table boundary detection. Section 4 elaborates the identification of the sparse lines. Section 5 demonstrates the experimental results. Section 6 is the related work. Conclusion and future work are included in section 7.

2 The Sparse-Line Property of Tables

Tables present structural data and relational information in a two-dimensional format and in a condensed fashion. Scientific researchers always use tables to concisely display their latest experimental results or statistical data. Other researchers can quickly obtain valuable insights by examining and citing tables. Tables have become an important information source for information retrieval. The demand for locating such information (table search) is increasing. To successfully get the table data from a PDF document, detecting the boundary of the table is a crucial phase.

Based on the observation, we notice that different lines in the same PDF document page have different internal space sizes, text densities, and the lengths. A document page contains at least one column and many journal/conference templates has two (e.g., ACM and IEEE template), or three even four columns. In a document, some lines have the similar lengths as the width of the document column, some are much longer (e.g., cross multiple document columns) and others are much shorter (e.g., the heading "1. Introduction" in our paper). From the internal space perspective, majority of the lines contains normal size of the space between two adjacent words but some lines have large spaces.

We define a line as a *sparse* line if it satisfies at least one of the following conditions:

- This line contains at least one large space gap between a pair of two consecutive words;

- The length of this line is much shorter than the width of the document column;

Non-sparse lines refers to the lines that satisfy neither condition. *Non-sparse* lines usually cover the following document components: the document title, the abstract, the body content paragraphs, etc. *Sparse* lines cover some other specific document components: tables, mathematical formulas, a part of texts in figures, most affiliations, references, etc. Since the majority of the lines in a document belongs to the non-sparse category, separating the document lines into such two categories according to the text internal space/density and filtering out the non-sparse lines become a fruitful preprocessing step for the table boundary detection. Such method has two advantages: 1) the sparse lines cover almost all the table content lines; 2) Narrowing down the table boundary to the sparse lines as early as possible can save large amount of time and effort that are spent on processing the non-sparse lines.

Some tables contain long cells that cross several table columns. In order to collect all the table cells, some researchers [6] make constraint on the number of such long lines within a table boundary. However, it is difficult to set a suitable value on it. If we set it too small, some parts of a table will be missed. If it is too large, some noisy lines will be included into the table boundary. In our method, such long cells will be treated as sparse lines and removed at the beginning. To decide whether merging the next sparse lines with the previous one and include them into the same table boundary, we only have to check the vertical space gaps between these two sparse lines. Different definitions of the "much shorter than" may generate different sparse line labeling results. We define the "much shorter than" as the half of the document column width.

We show a snapshot of a PDF document page in Figure 1(a) as an example. In Figure 1(b), we highlight all the sparse lines in red rectangles. Apparently, all the table body content lines are labeled as sparse lines. In Figure 1(b), there are four sparse lines that are not located within the table boundary: the heading lines "Conclusions" and "consumer products", and the document content line "proposed method" and "dilution used in the analyses.". We label them as sparse lines because they satisfy the second condition: the length of this line is much shorter than the width of the document column. Since such short-length lines also happen in some table rows with only one filled cell, we consider them in case of omitting the potential table lines. Such noisy non-table sparse lines are very few because they usually only exist at the headings or the last line of a paragraph.

[4]http://www.pdflib.com/products/tet/

[5]http://snowtide.com/PDFTextStream

[6]http://ieg.ifs.tuwien.ac.at/projects/pdf2table/

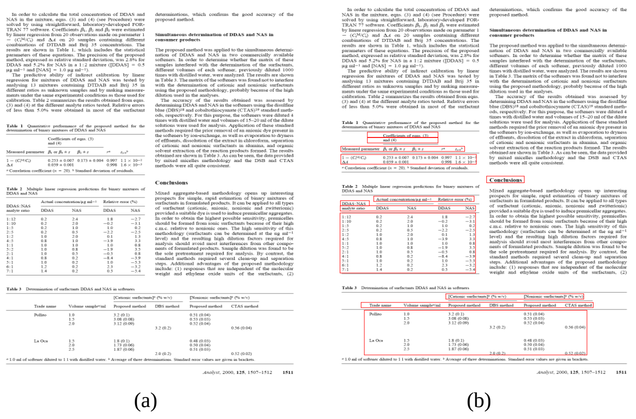

Figure 1. The sparse lines in a PDF page

In addition, the short length restriction also reduce the frequency heavily. We can easily get rid of them based on the coordinate information later.

3 The Text Object in PDF

3.1 Text Tables vs. Image Tables

Each PDF document can be viewed as a sequence of pages, which in turn can be recursively decomposed into a series of components, such as text, graphics, and images. The corresponding objects to those components are text objects, image objects, path objects, etc, which listed by PDF document content stream. Path objects are referred to as vector graphics objects or drawings and paintings composed by lines, curves, and rectangles. They are the building blocks for the graphical illustrations, such as bar charts, pie charts and logos are often just a fraction of the whole figure illustrations e.g. one bar in a bar chart. Page objects in PDF documents don't reflect nor are related to the logical structure or logical components of the document. The text object provides information of characters in the text as well as many important attributes, such as the X-Y coordinates, the font size, the font type, the color, the spacing, the orientation, etc.

Most of the existing research to discover the logical components of a document focus on analyzing most if not all of these page objects. For example, regrouping all the objects to form the image is a traditional task for document analysis system. However, the object overlapping problem happens frequently in these researches and the researchers have to make more effort to segment these objects from each other first. In addition, even when such objects or structures are identified, they are still too high level to fulfill specific goals, such as the table detection.

For most table related applications (e.g., table data extraction and table searching), the majority research interests are focused on the text (the table content), instead of the borderlines. We randomly examine thousands of the PDFs in the computer science, chemistry, biology, and archeology fields and notice that most table contents consist of texts while only few tables contain images. Based on this observation, we classify all tables into two categories: the *text table* and the *image table*.

Text tables refer to those tables that all parts are composed of texts. Image tables refer to those tables that are image themselves or contain images in some cells. All the three tables in Figure1 are text tables. Figure2(a) displays an example of an image table that all the cells are filled with images and Figure2(b) displays an example of an image table that the image cells and the text cells mingle with each other. Within randomly selected 100 tables in each fields spanning the period from 1980 to 2007, the number of the image tables are 0, 3, 8, and 5 respectively.

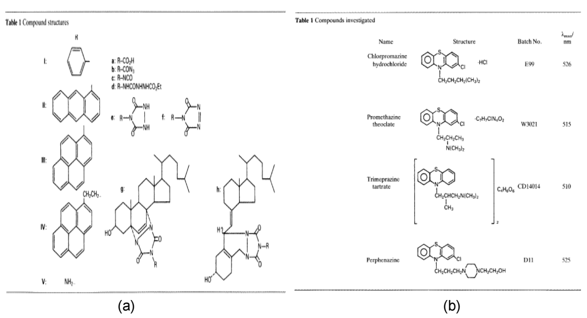
Table 1 Compound structures
Table 1 Compounds investigated
Name
Structure
Batch No.
Chloropromazine hydrochloride
Promethazine theoclate
Trimeprazine tartrate
Perphenazine
(a)
(b)

Figure 2. Two examples of the image tables

3.2 Text Extraction

In our work, we directly analyze PDF documents instead of converting them to HTML or Image file because of the following five reasons: 1) the existing text extracting tools can help us to obtain the information of texts; 2) converting PDF to HTML does not provide any additional help in detecting the tables than directly analyzing text information from PDF because it is impossible to know where the tables are in advance. 3) additional work is needed to process the converted HTML or image files; 4) directly analyzing PDF document can provide more accurate results than document image analysis and OCR; 5) the results are not affected by the contrast of the text, the background and the text overlay.

Because most tables are composed of texts, the text extracting tools, which only provide the very low level information (characters, words, coordinates, etc) without the structure information, is enough for our goal. Many PDF converters are available off the shelf (Xpdf, PDF2TEXT, PDFBOX, Text Extracting tool, PDFTEXTSTREAM, etc). The information obtained with the help of these text extraction tools can be divided into two categories: the *text* *content* and the *text style*. The text content refers the text strings; The text style includes the corresponding text attributes: the font, the size, line spacing and color, etc;

The text streams extracted from PDF files can correspond to various objects: a character, a partial word, a complete word, a line, etc. In addition, the order of these text streams does not always correspond to the reading order. A word reconstruction and a reading order resorting steps are necessary in order to correctly extract the text from a PDF file. In the next section, we describe the word reconstruction component. The details of our text sequence resorting algorithm is beyond the topic of this paper and will be elaborated in a subsequent paper. In this paper, after the word reconstruction and the sparse line detection, it is assumed that the text sequence is correct as a matter of course.

4 Sparse Line Detection

Before classifying a document line as a sparse/non-sparse line, we have to construct the lines first. We adopt a bottom-up approach to start the procession from the character level in PDF documents. The first step is still the extraction of the native PDF text information. Then we group the characters into words and lines based on the textual features and the reading order.

Adobe's Acrobat word-finder provides the coordinate of the four corners of the quad(s) of the word. The PDFlib Text Extraction Toolkit (TET) also provides the function to extract the text in the different levels (characters, words, paragraphs, etc.). However, it only provides the content instead of other style information in all the levels except the character level. If we want to do some further work, content itself is usually not enough. We have to calculate the corresponding coordinates for the higher levels by merging the characters.

Initially, for each PDF document, the text extraction tools strip out the text information from the original PDF source file character by character through analyzing the *text operators*[7] and the related glyph information. Similar to Xpdf library, we reconstruct these characters into words then lines with the aid of their position information and saves the results into a Document Content File in the TXT format. To convert characters/words into words/lines, we adopt some heuristics based on the distance between characters/words. For each document page, we analyzes the text information and label each line as a sparse/non-sparse line according to their internal word relative position information and width. The main unique place of our method is that we only analyze the coordinate information. Font information, the frequently adopted parameter, is abandoned here because it is not reliable.

[7]PDF Reference Fifth Edition, Version 1.6

Table 1. The thresholds we adopted for word reconstruction

Parameters	Definition
α	the vertical distance between two top Y-axis values: $alpha = Y_{i+1} - Y_i$
β	the vertical distance between two bottom Y-axis values: $beta = Y^{,}_{i+1} - Y^{,}_i$
γ	the horizontal distance between these two characters: $\gamma = X_{i+1} - X^{,}_i$
δ	the vertical distance of two characters
θ	the maximal width of the space with a word
η	the maximum vertical distance between two characters in a same line

4.1 Characters $\rightarrow$ words

Formally, we define a document as a set of pages D = $\cup_{k=1}^{n}(P_k)$, where n is the total page number. Each page P_k, can be denoted as an aggregation of characters $C \in \{Character\}$. c_i and c_{i+1} are a pair adjacent (no other character exists between them) characters. Initially, we get the coordinate of the first character c_0 in a document page. All the characters in C share a common set of attributes $\{([X, X^{,}], [Y, Y^{,}], W, H, F, T)\}$, where $[X, Y]$ is the pair of coordinators of the upper-left corner of the character while $[X^{,}, Y^{,}]$ is the coordinates of the bottom-right corner of the character. The original point of the X-Y axes is the left-bottom corner of a document page. W/H denotes the width/height of the component, F is the font size, and T is the text. Figure 3 shows the coordinates of an example character. In this paper, we do not use the font size because in many journals and archives, the font information (the font type and the font size) is not so standard as we imaged. Considering such disordered information will incur error to the final results. Therefore, we only analyze the coordinates because W and H are also implicated in it.

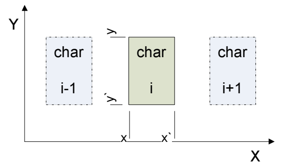

Figure 3. The coordinates of a character in a PDF document page.

Since the character C is the fundamental component of a document, other components can be constructed recursively from it. For example, a document *page* P_k can be denoted as an aggregation of *words* $W = \{w_j | w_j = ([X_{w_j}, Y_{w_j}]), ([X^{,}_{w_j}, Y^{,}_{w_j}]), W_{w_j}, H_{w_j}, F_{w_j}, T_{w_j}\}$. A document word w_j is equal to $\cup_{i=1}^{m} c_i$, where m is the total number of characters in the word w_j.

Figure 4 enumerates all the relative positions of a pair of adjacent characters c_i and c_{i+1}. Their coordinates are $([X_i, X_i^{,}], [Y_i, Y_i^{,}])$ and $([X_{i+1}, X_{i+1}^{,}], [Y_{i+1}, Y_{i+1}^{,})$ respectively. For the word reconstruction, we define several parameters and thresholds, which are listed in Table 2:

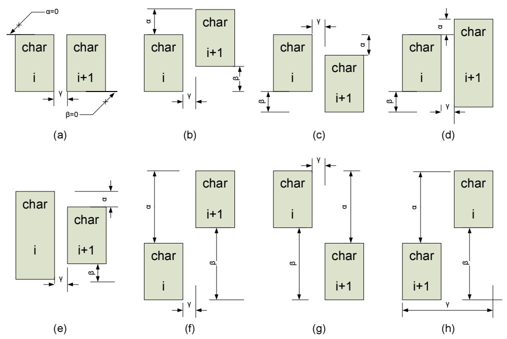

Figure 4. The coordinates of the example cases of the character pairs

Figure 4 (a) presents a common character pair in the same line. $Y_{i+1} = Y_i(\alpha = 0)$ and $Y_{i+1}^{,} = Y_i^{,}(\beta = 0)$. If γ is smaller than a given threshold θ, the second character c_{i+1} can merge with c_i into a same word. Otherwise, we treat c_i as the last character of the current word and c_{i+1} as the starting character of a new word.

Figure 4 (b) – (e) display several examples of the same-line character neighbors with partial vertical overlaps. Superscript is a typical case of Figure 4 (b) while subscript is a typical case of Figure 4 (c). Figure 4 (d) and (e) show the font size changing within a document line. All these character pairs are also same-line characters. Every case has to satisfy some fixed conditions as follows: $Y_{i+1} \geq Y_i \geq Y_{i+1}^{,} \geq Y_i^{,}$ (Figure 4 (b)), $Y_i \geq Y_{i+1} \geq Y_i^{,} \geq Y_{i+1}^{,}$ (Figure 4 (c)), $Y_{i+1} \geq Y_i \geq Y_i^{,} \geq Y_{i+1}^{,}$ (Figure 4 (d)), and $Y_i \geq Y_{i+1} \geq Y_{i+1}^{,} \geq Y_i^{,}$ (Figure 4 (e)). For these cases, we decide whether c_i and c_{i+1} go into the same word or not, by comparing γ with the same threshold θ;

To analyze the character pairs in Figure 4 (f) – (h), we introduce another threshold η: *the maximum vertical distance between two characters in a same document line*. In $4(f)$, the fixed constraint is $\delta = (Y_{i+1}^{,} - Y_i) > 0$. In Figure $4(g)$ and (h), the fixed constraint is $\delta = (Y_i^{,} - Y_{i+1}) > 0$. If $\delta > \eta$, C_i and C_{i+1} will belong to different lines. Otherwise, we treat them as the character neighbors in a same line and decide whether they go to the a same word. Starting a new document column in a page is a typical examples with large γ for case f, and starting the next line is a typical examples with large but minus γ for case h. Using the Table 1 in Figure1 as the example, we show the merged words in Figure 5. Each red rectangle refers an independent word.

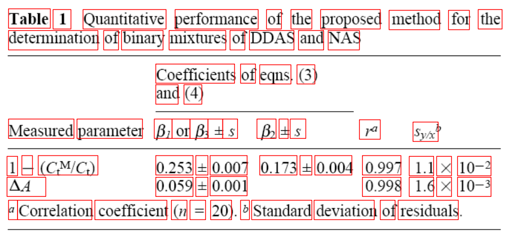

Table 1 Quantitative performance of the proposed method for the determination of binary mixtures of DDAS and NAS

Measured parameter	Coefficients of eqns. (3) and (4)			
	β_1 or $\beta_3 \pm s$	$\beta_2 \pm s$	r^a	$s_{y/x}{}^b$
$1 - (C_t^M/C_t)$	0.253 ± 0.007	0.173 ± 0.004	0.997	1.1×10^{-2}
ΔA	0.059 ± 0.001		0.998	1.6×10^{-3}

[a] Correlation coefficient (n = 20). [b] Standard deviation of residuals.

Figure 5. The merged words in a table after the character → word phase

4.2 Sparse line detection

Now a document page p_k can be denoted as an aggregation of words W. Each word is composed of a set of character sequences $\cup_{k=1}^{B}(c_k)$, where $B > 0$ is the word length in terms of characters. w_i and w_{i+1} are a pair of adjacent (no other object exists between them) words. Initially, we get the coordinate of the first word w_0 in a document page. All the characters in W share the same coordinate attributes as that of the character level $([X, X^{,}], [Y, Y^{,}])$. Similar to the characters, we can also treat words as rectangle objects in a document page. The coordinate nature of characters in the previous section is also applicable to the words. For a pair of word neighbors, w_i and w_{i+1}, the possible relative locations are same of the cases listed in Figure 4.

To detect the sparse lines, we use the same combining method in Section 5.1. We believe that in the non-sparse lines, all the words can be merged together into one piece. Sparse lines refer those lines that contain more than one text pieces with the same Y-axis after the combination. Using the concept in Section 5.1, we should treat the *word* here as the *character* there and treat the *text piece* here as the *word* there. The parameters and the thresholds in Table 2 can be reused with only the value resets of γ and θ. Because of the space limitation, we do not repeat the process here.

After the combination, we check the number of text pieces in each Y-axis along the sequence generated by the text extraction tools, if the number is larger than one, we label this line as the sparse line. If the number is one but there are only one word in this line, we also treat it as a sparse line. Still using the Table 1 in Figure1 as the example, we show the merged lines in Figure 6. For all the eight lines, the number of text pieces are *1, 1, 1, 1, 5, 5, 4,* and *1* respectively. We treat line 5, 6, and 7 are sparse lines because they contain more than one text piece. We also treat the line 3 and 4 as sparse lines because of their small width.

4.3 The effect of the Font information

Some researchers also use the font information. However, we think it is not so reliable as stated. In many PDF

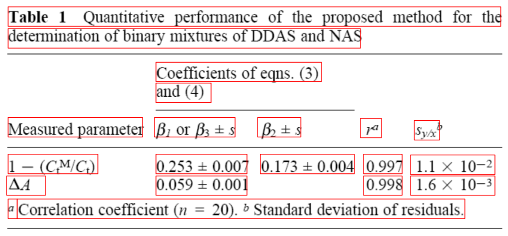

Table 1 Quantitative performance of the proposed method for the determination of binary mixtures of DDAS and NAS

	Coefficients of eqns. (3) and (4)			
Measured parameter	β_1 or $\beta_3 \pm s$	$\beta_2 \pm s$	r^a	$s_{y/x}{}^b$
$1 - (C_t^M/C_t)$	0.253 ± 0.007	0.173 ± 0.004	0.997	1.1×10^{-2}
ΔA	0.059 ± 0.001		0.998	1.6×10^{-3}

[a] Correlation coefficient ($n = 20$). [b] Standard deviation of residuals.

Figure 6. The merged lines in a table

documents, from the human eye view, it seems that the texts in a same line have unified font types and sizes (For example, many CS conferences). However, when we process them using the text extraction tools, we find that the font size may have minor inconsistence. Such inconsistence may incur troubles in the word/line merging step. Even though the font changing is large enough to be observed, the solely font information is not enough. The coordinate information is still needed to judge the relative position of different text pieces. Considering the additional font information can not generate significant improvement. Our experimental results confirm this point of view.

Some researchers may argue that the changing of the font information can facilitate the detection of the document component switch. For example, the font size of a table is smaller than the font size of the document body content texts. The font size of the document title is usually larger than the font size of the affiliation component. We call this phenomenon as the *"document component font changing"* property. We agree that such font information may have a positive impact on the table boundary detection, if such text inconsistence exists and we know them in advance. In this paper, we do not consider the font information because of two reasons:

- the *"document component font changing"* property is not strictly adopted. Many journal/conference proceedings do not follow this property. For example, DAS conference has a table type difference between the document main text and the figure/table captions (both are 10-point), but the font sizes are same: the main text should be in 10-point times and the figure and table captions should be 1o-point boldface Helvetica. However, some conferences adopt the consistant font for main text, figure, and tables.

- Even though the *"document component font changing"* property exists, in order to use such specific font information to detect the table boundary, we have to know the details in advance. However, it is impossible in many cases with the only input of the PDF document.

4.4 Detecting the table boundary

After the sparse line detection, we can easily detect the table boundary by combining the sparse lines with the table keywords. Here we define the main table content rows as the table boundary, which does not have to include the table caption and the footnote. In order to enhance the performance of the table starting location detection, we consider the keyword information. Of course, we can directly detect the table boundary by detecting the tabular structure within the sparse line areas.

In our method, we define a keyword list, which lists all the possible starting keywords of table captions, such as "Table, TABLE, Form, FORM," etc. Most tables have one of these keywords in their captions. If more than one tables are displayed together, the keyword is very useful to separate the tables from each other. Once we detect a line (not only the sparse line) starting with a keyword, we treat it as a table caption candidate. Then we check the sparse lines after the caption and merge them into a sparse area according to the vertical distance of the line gaps. Such sparse-line areas are the detected table boundary. Because the texts within the detected table boundary will be analyzed carefully in the later table structure decomposition phase, we treat recall more importantly than precision here.

5 Experiments and Results

In this section, we demonstrate the experimental results of evaluating our table boundary detection. Our experiments can be divided into two parts: the evaluation of the sparse line detection algorithm based on the coordinate information, and the evaluation of the table boundary detection, with/without font information. Before describing the experimental details, we first discuss document collection.

We focus on tables in scientific documents in PDF. the PDF document collection comes from three sources: 1) scientific digital libraries (Royal Chemistry Society[8]), Citeseer[9]), and archeology[10] in three fields: chemistry, computer science and archeology. The size of each PDF repository exceeds $100,000$, $1,000$ and $8,000$ respectively. All the documents span the years 1950 to 2007.

5.1 Experimental Results of sparse line detection

We perform a five-user study to evaluate the quality of the sparse line detection, according to the coordinates. Each user checks the detected sparse lines in 20 randomly selected PDF document pages. The evaluation metrics are

[8]http://www.rsc.org/
[9]http://citeseer.ist.psu.edu/
[10]http://www.saa.org/publications/AmAntiq/AmAntiq.html

precision and recall. The total number of the testing PDF pages is 300. Given the number of true sparse lines extracted by our method A, the number of true positive sparse lines but overlooked B, and the number of true negative non-sparse lines that is misidentified as sparse lines C, the *Precision* is $\frac{A}{A+C}$, and the *Recall* is $\frac{A}{A+B}$.

Table 2. The performance evaluation of the sparse line detection

Field	Chemistry	Archeology	CS
The Number of PDF pages	100	100	100
Recall of sparse line detection	99.9	100	100
Precision of sparse line detection	99.6	99.2	98.7

We misidentify some non-sparse lines as sparse lines because some document lines have large internal spaces. Several factors cause the large spaces: 1) the setting of the threshold γ; 2) the text missing problem of the text extraction tool.

Some tables have long cells. Because of the limited space, such tables have crowd columns with very small spaces between the adjacent table columns. Such small column space is the main reason for the missed sparse lines in the chemistry PDF documents.

Within a same word, different characters have the same font properties. Within a same line, although there may exist some font diversity among different words (e.g., the superscript, the subscript, or mathematical symbols), such font difference is not used in our method as the rule to decide whether merge the next word into the same text piece or not. Therefore, the font information does not affect the performance of the sparse line detection.

5.2 Experimental Results of the table boundary detection based on the sparse lines

Table 4 displays our experimental results of the table boundary detection based on the detected sparse lines in the same 300 PDF document pages. In this part, we still use the precision and recall as the evaluation metrics. Given the number of true table boundaries extracted by our method A, the number of true positive table boundaries but overlooked B, and the number of non-table areas misidentified as table areas C.

In order to prove the importance of the coordinates and the effect of the font information, we also implemented the same evaluation by combining the font information. If the next lines have different font type or font size, we think that the current document component stops at the previous line and the next line starts a new component. After checking the 300 PDF pages, we notice that only 66% pages have font changes when a table begins.

It is not surprising to find that the font information can not improve the performance of the table boundary detection. In some cases, the performance is even worse. The first reason is that the font changing is also applicable to other document components, e.g., figures and references. Considering this factor may not only facilitate the detection of the table beginning places, but also incur more works and noisy results by examining other components. The second reason is that in many old documents, the font information is not standard and minor changes happen everywhere.

We may include some external sparse lines if they are close to a table boundary. Such noisy lines can be easily removed in later table structure decomposition step. If a table has a long single-cell row, such row is usually be filtered out because we label it as a non-sparse line. However, because we treat all the texts (not only the sparse lines) within the detected table boundary as table contents, such missed lines can be easily retrieved back.

Table 3. The performance evaluation of the table boundary detection based on the detected sparse lines

	Chemistry	Archeology	CS
Recall (without font)	97.6	96.3	95.5
Recall (with font)	97.1	90.8	95.5
Precision (without font)	98.9	98.2	98.8
Precision (with font)	99.1	98.2	98.8

Some tables are labeled using other keywords, especially the documents in computer science field, which name the tables as "Figure," which are confusing with real figures. To avoid real figures and to keep high efficiency, we overlooks such "wrongly" labeled tables in current stage. However, this problem can be overcame by heuristics that identify the grid-structure cells among the sparse line set. Moreover, the performance of the text extraction tools directly affects the table extraction results. Currently, *PDFBOX* and *TET* are used to fetch all the text information in the documents. Characters missing and space insertions are two typical inherited errors, which may tamper with the performance. This problem is orthogonal to our work and we hope that these problems will be addressed independently by the designers of the text extraction tools.

6 Related Work

Researchers in the automatic table extraction field largely focus on analyzing the table structure in a specific document media. Zanibbi [11] provides a survey with detailed description of each method. All the methods can be divided into three categories: pre-defined layout based [9], heuristics based [6][8][10][12], and statistical based. Pre-defined layout based algorithms usually work well for one domain, but is difficult to extend. Heuristics based methods need a complex post-processing and the performance relies largely on the choice of features and the quality of training data. Most approaches described so far utilize purely geometric features (e.g. pixel distribution, line-art, white streams) to determine the logical structure of the table, and different document mediums require different process methodologies: OCR [4], X-Y cut [7], tag classification and keyword searching [1][3][13] etc. In the past two decades, a good number of researches have been done to discover the document layout by converting the PDFs to image files. However, the image analysis step can introduce noise (e.g., some text may not be recognized or some images may not be correctly recognized). In addition,because of the limited information in the bitmap images, most of them only work on some specific document types with minimal object overlap: e.g., business letters, technical journals, and newspapers. Some researchers combine the traditional layout analysis on images with low-level content extracted from the PDF file.

Chao et.al. reported their work on extract the layout and content from PDF documents. Hadjar et al. have developed a tool for extracting the structures from PDF documents. They believe that to discover the logical components of a document, we need to analyze all/most of the page objects such as text objects, image objects, path objects, etc, which are listed by PDF document content stream. However, the object overlapping problem happens frequently, if we analyze all the objects, We have to spend more effort to segment these objects from each other first. In addition, even we identified such objects/structures, they are still too high-level to fulfill many special goals, e.g., detecting the tables, figures, mathematical formulas, footnotes, references, etc.

Instead of converting the PDF documents into other types of media (e.g., image or HTML) and then applying the existing techniques, we process PDF documents directly from the text level. In this paper, we propose a method that relies solely on the PDF extracted content, not longer requiring the conversion to any other document medium and apply any further processing methods.

7. Conclusions

In this paper, we propose a novel method to detect the table boundary. Because most tables are text-based, we claim that the text object of PDF provides enough information for table detection. Within the text object, we believe that the font size is not so reliable as other work stated. Based on the sparse-line nature of tables, we propose a fast but effective method to detect the table boundary by only processing the sparse lines in a document page. Processing the sparse lines solely can also improve the performance of the text sequence resorting problem. Combining different keywords, this method is applicable tror o detect other document components, e.g., figures.

References

[1] W. G. B. Krupl, M. Herzog. Using visual cues for extraction of tabular data from arbitrary html documents. In *In Proc. of the 14th Int'l Conf. on World Wide Web*, pages 1000–1001, 2005.

[2] H. Chao and J. Fan. Layout and content extraction for pdf documents. pages 213–224, 2004.

[3] S. T. H. Chen and J. Tsai. Mining tables from large scale html texts. In *In Proc. 18th Int'l Conf. Computational Liguistics, Saarbrucken, Germany*, 2000.

[4] X. W. W. B. D. Pinto, A. McCallum. Table extraction using conditional random fields. In *In proceeding of Proceedings of the 26th ACM SIGIR, Toronto, Canada*, July 2003.

[5] A. N. Expert. *A Book He Wrote*. His Publisher, Erewhon, NC, 1999.

[6] J. T. K. H.T. Ng, C. Y. Lim. Learning to recognize tables in free text. In *In Proc. of the 37th Annual Meeting of the Association of Computational Linguistics on Computational Linguistics*, pages 443–450, 1999.

[7] R. H. J. Ha and I. Philips. Recursive x-y cut using bounding boxes of connected components. In *In Proc. Third Int'l Conf. Document Analysis and Recognition*, pages 952–955, 1955.

[8] N. G. J. Shin. Table recognition and evaluation. In *In Proc. of the Class of 2005 Senior Conf., Computer Science Department, Swarthmore College*, pages 8–13, 2005.

[9] T. W. J.H. Shamilian, H.S. Baird. A retargetable table reader. In *In Proc. of the 4th Int'l Conf. on Document Analysis and Recognition*, pages 158–163, 1997.

[10] T. G. Kieninger. Table structure recognition based on robust block segmentation. In *In Proc. Document Recognition V, SPIE, volume 3305*, pages 22–32, January 1998.

[11] D. B. R. Zanibbi and J. Cordy. A survey of table recognition: Models, observations, transformations, and inferences. In *Int'l J. Document Analysis and Recognition, Vol. 7, No.1*, pages 1–16, 2004.

[12] A. D. T. Kieninger. Applying the t-rec table recognition system to the business letter domain. In *In Proc. of the 6th Int'l Conf. on Document Analysis and Recognition*, pages 518–522, September 2001.

[13] J. H. Y. Wang. Detecting tables in html documents. In *In Proc. of the 5th IAPR Int'l Workshop on Document Analysis Systems, Princeton, NJ*, 2002.