2009 10th International Conference on Document Analysis and Recognition

Improving the Table Boundary Detection in PDFs by Fixing the Sequence Error of the Sparse Lines

Ying Liu, Kun Bai, Prasenjit Mitra, C. Lee Giles
College of Information Sciences and Technology
The Penn State University, University Park, PA, 16803
{yliu, kbai, pmitra, giles}@ist.psu.edu

Abstract

As the rapid growth of PDF documents, recognizing the document structure and components are useful for document storage, classification and retrieval. Table, a ubiquitous document component, becomes an important information source. Accurately detecting the table boundary plays a crucial role for many applications, e.g., the increasing demand on the table data search. Rather than converting PDFs to image or HTML and then processing with other techniques (e.g., OCR), extracting and analyzing texts from PDFs directly is easy and accurate. However, text extraction tools face a common problem: text sequence error. In this paper, we propose two algorithms to recover the sequence of extracted sparse lines, which improve the table content collection. The experimental results show the comparison of the performance of both algorithms, and demonstrate the effectiveness of text sequence recovering for the table boundary detection.

1. Introduction

Portable document format (PDF) is a widely used document format. Although many researchers analyze PDF documents by converting them to other formats (e.g., image, html), automatically identifying the PDF document logical structures information and document components (e.g., figures, tables, etc) are still challenging problems [2] because of three main reasons: 1) extracted texts from PDF files are non-tagged; 2) wrong text sequences are generated by the text extraction tools; 3) new noises are caused by necessary tools (e.g., OCR) when converting the PDFs into other format (e.g., image).

Tables, as a specific document component, are ubiquitous everywhere. To thoroughly analyzing the table content, locating the table boundary in a document is the first and crucial step for consequent applications (e.g., the table search). Because most of the tables in scientific documents are text-based tables, solely analyzing the text objects in a PDF file can achieve good performance in terms of saving efforts on other objects. However, existing text extraction tools face a common problem: the extracted text pieces follow a different sequence from its original appearance in PDF files. Although the wrong sequence does not affect the PDF documents displaying for browsing purpose, it can severely interfere the document content analyzing works that rely on the information of the relative location and sequence among the text pieces. Table boundary detection is a typical example. In order to improve the accuracy of the table boundary detection, we need to recover the extracted text sequences to the orders that comply with the human's reading habit. The sequence errors can be classified into two types: *within-table errors* and *beyond-table errors*. In this paper, we propose two text sequence recovering algorithms to fix them. Our algorithms start with the detected sparse lines in each PDF page and detect the table boundaries by analyzing the sparse areas and the caption information. The detail of the sparse line detection is out of the scope of this paper. If interested, the details can be found in [3] and [4].

The rest of the paper is organized as follows. Section 2 gives the definition for main terms used in this paper. Section 3 elaborates two text sequence error categories. Section 4 introduces two text sequence recovering algorithms and how to detect the table boundary based on the recovered sequences. Section 5 demonstrates the experimental results. Conclusion and future work are included in section 6.

2 Term Definitions

Free online dictionary[1] defines a table as *an orderly arrangement of data, especially one in which the data are arranged in columns and rows in an essentially rectangular form.* In addition, we add an additional restriction on the definition of tables in scientific documents: each table has a caption that starts with a keyword (e.g., "Table" or "TABLE"). With this restriction, the tables can be understood as genuine tables [6]. The table boundary includes all the

[1]http://www.thefreedictionary.com/table

978-0-7695-3725-2/09 $25.00 © 2009 IEEE
DOI 10.1109/ICDAR.2009.138

parts in the Wang's model [5]. Both table caption and footnote are not included.

Different lines in the same page have different internal *space size*, *text density*, and *length*. A document page contains at least one column and many have two (e.g., ACM and IEEE conference formats), or three even four columns. The length of lines can be equal to, longer (e.g., cross-column lines), or shorter (e.g., some headings) than the width of the document column. Because single-column documents are easy to process, in this paper, we focus on the most popular yet challenging digital document template: double-column formatting. From the internal space perspective, the majority of the lines contain normal space size while some lines have large space size, between two adjacent words. We define a line as a *sparse* line if it satisfies at least one of the following conditions: 1) This line contains at least one large space gap between a pair of consecutive words within the line; 2) The length of this line is much shorter than a threshold *ll*; Note, different definitions of "large" and the threshold *ll* generate different results. Based on the observation and the statistical results on thousands of table examples, in this paper we set the "large" as the double of the average space between words in document body content lines, and *ll* as the two third of the document column width;

Non-sparse lines, which satisfy neither of the conditions, usually occur in the following document components: long document titles, the abstracts, the body content paragraphs, etc. *Sparse* lines cover other specific document components: tables, texts in figures, mathematical formulas, document affiliations, and references, etc. Since the majority lines in a document are non-sparse lines, filtering them out as early as possible is an effective preprocessing step for the table boundary detection because we can save a substantial amount of time and effort and narrow down the table boundary to the sparse lines easily. The detection and filtering work is out the scope of this paper. If interested, the details can be found in [3] [4]. Our text sequence recovering algorithm starts with the detected sparse lines. Figure 1 shows in a document page with a wide table while Figure 2 shows the filtered sparse lines in another document page with two parallel tables.

3 The text sequence problem

Given a PDF document page, the human reading sequence is usually top-down, left-to-right, and line by line, column by column. Particularly, some document components in the page may cross multiple document columns, such as the document title, the affiliation, some wide tables and figures. Although the text extraction tools can extract all the text information from a PDF page, the sequence of the exported texts may not reflect the human reading ordering. The text sequence error is a common problem shared by existing text extraction tools (e.g., PDF2TEXT,

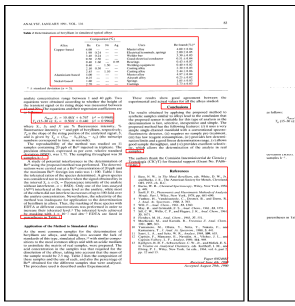


Figure 1. A document page with a cross-column table (left) and its sparse lines (right)

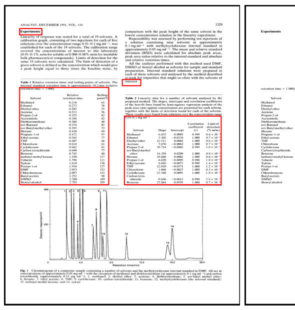
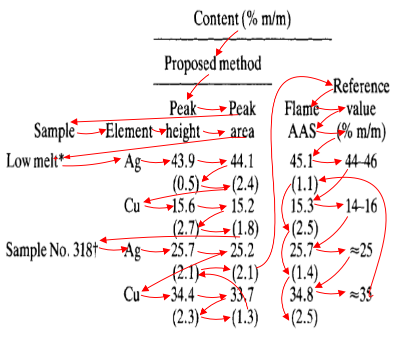


Figure 2. A document page with two parallel tables (left) and its sparse lines (right)

PDFBOX, TET, PDFTextStream). Such text sequence error happens frequently in the table contents, which may generates serious wrong results during the table detection, such as segmenting a table into several pieces, wrong column/row information, and omitting cells/whole tables etc. Some of the errors can be fixed with complex post-processing while others can not. According to the scope of the text sequence errors, we category them into two classes: *inside-table* text sequence error and *beyond-table* text sequence error.

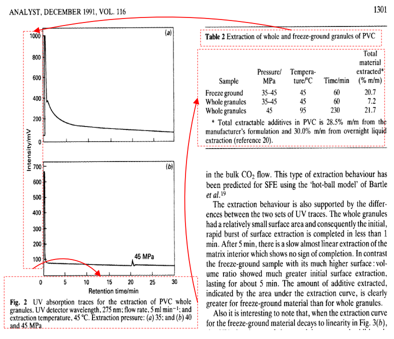
Figure *3a* provides an example of the *inside-table* text sequence error. The text extraction tools export 33 sparse lines in such a sequence: *"Content (% m/m)", "Proposed method", "Peak Peak", "Sample Element height area", "Low melt Ag 43.9 44.1", "0.5 2.4", "Cu 15.6 15.2", "2.7 1.8", "Sample No.318+ Ag 25.7 25.2", "Cu 34.4 33.7", "2.3 1.3", "2.1 2.1", "Reference", "Flame value", "AAS (%m/m)", "45.1 44.46", "15.3 14-16", "25.7 ≈25", "34.8 ≈35", "1.1", "2.5", "1.4", "2.5"*. The sequence is: the first four columns, then the last two columns. Moreover, within the same column, the ordering of rows does not exactly follow the top-down manner. In this type, the errors happen only within the table boundary and no outside text interrupts them. No matter how disordered the sequence is, all the contents of a table come together as a whole.

In *beyond-table* text sequence error, the table lines are partitioned into at least two parts by non-table lines. The text extraction tools usually export a part of the table con-



		Content (% m/m)			
		Proposed method			
		Peak	Peak	Flame	Reference value
Sample	Element	height	area	AAS	(% m/m)
Low melt*	Ag	43.9	44.1	45.1	44–46
		(0.5)	(2.4)	(1.1)	
	Cu	15.6	15.2	15.3	14–16
		(2.7)	(1.8)	(2.5)	
Sample No. 318†	Ag	25.7	25.2	25.7	≈25
		(2.1)	(2.1)	(1.4)	
	Cu	34.4	33.7	34.8	≈35
		(2.3)	(1.3)	(2.5)	

(a)



ANALYST, DECEMBER 1991, VOL. 116 1301

Fig. 2 UV absorption traces for the extraction of PVC whole granules. UV detector wavelength, 275 nm; flow rate, 5 ml min^{-1}; and extraction temperature, 45 °C. Extraction pressure: (*a*) 35; and (*b*) 40 and 45 MPa.

Table 2 Extraction of whole and freeze-ground granules of PVC

Sample	Pressure/ MPa	Tempera- ture/°C	Time/min	Total material extracted* (% m/m)
Freeze ground	35–45	45	60	20.7
Whole granules	35–45	45	60	7.2
Whole granules	45	95	230	21.7

* Total extractable additives in PVC is 28.5% m/m from the manufacturer's formulation and 30.0% m/m from overnight liquid extraction (reference 20).

in the bulk CO_2 flow. This type of extraction behaviour has been predicted for SFE using the 'hot-ball model' of Bartle *et al.*[19]

The extraction behaviour is also supported by the differences between the two sets of UV traces. The whole granules had a relatively small surface area and consequently the initial, rapid burst of surface extraction is completed in less than 1 min. After 5 min, there is a slow almost linear extraction of the matrix interior which shows no sign of completion. In contrast the freeze-ground sample with its much higher surface : volume ratio showed much greater initial surface extraction, lasting for about 5 min. The amount of additive extracted, indicated by the area under the extraction curve, is clearly greater for freeze-ground material than for whole granules.

Also it is interesting to note that, when the extraction curve for the freeze-ground material decays to linearity in Fig. 3(*b*),

(b)

Figure 3. Two examples of the text sequence errors

tent first, jump to another document area and process it, and then come back to finish the rest of the table sections. Such alternation happens at least once. Figure 3b shows an example: the text extraction tools output the table caption first on the second document column, then jump to the first document column on the left and process some scattered texts in the figures. After telling the figure caption, they come back to the table area. Sometimes, the tools extract several table columns then jump to other areas (e.g., the following regular document contents) to finish several lines or paragraphs, then return to the table data again. With such *beyond-table* text sequence errors, collecting all the table cells is a challenging task because it is difficult to predict how far the distances are among different table parts.

In order to accurately collect the table cells from a page and recover the sequence, resorting the texts in the page is one approach. B. Yildiz et. al mentioned the similar idea in [7] but without any detail about the methodology and the performance. Other proposed approaches [1] exploit geometric or typographic features of the page objects, and go further in exploiting the content of object. X-Y cut approach is a representing method. However, its performance is not satisfying. Moreover, these proposed works need to process both sparse lines and non-sparse lines. The more non-sparse lines involve, the higher possibility errors incur.

4 Text sequence recovering algorithms

To fix the text sequence error problem, we propose two methods to resort the detected sparse lines within a given area. For the inside-table text sequence errors, the area is the table itself. For the beyond-table text sequence errors, the area is the whole page. Some pages have multiple document columns, whether considering such information or not may generate different results for different cases. In this paper, we try both of them and analyze their advantages/disadvantages.

4.1 Algorithm 1: Considering the document column information

The recovering procedure includes two parts: the *cross-column resorting*, and the *within-column resorting*. Here the *column* refers to the document column instead of the table column. How to get the document column information is out of scope of this paper. We adopt an easy but effective method by calculating the average length of the non-sparse lines then comparing with the document width. For the *inside-table text sequence disordering*, we only implement the *within-column resorting*. For the *beyond-table text sequence disordering*, we have to implement both resortings.

In the *cross-column resorting*, SL denotes the set of all the sparse lines in a document page: $SL = \bigcup_{i=1}^{m}(l_i)$. For a document column CO_k, its beginning and ending X-axes values are X_{co_k} and X'_{co_k}. Each column has a vector v to store its sparse lines. Initially, the vector is empty. For each sparse line, we compare its X-axis coordinates with document columns and append it in the matched vector v, if applicable. Some sparse lines may cross several document columns. Such lines usually are lines in cross-column tables/figures or long document titles or affiliations, etc. We record such lines in another vector v_{cc}.

In the *within-column resorting* part, the inputs include the filled vector V and V_{cc}. The output is a new vector V_{sorted}, which stores all sorted sparse lines. For each document column CO_k, we get all the Y-axis values from the sparse lines in vector v_k. Among these Y-axis values, we identify all the non-duplicated values and sort them, and store them in a new vector V_y. With these Y-axis values, we sort all sparse lines in vector v_k accordingly. This method works well for the tables that are located within one document column (e.g., the table in Figure 3b). However, for the wide tables that cross more than one document column (see Figure 1a), this algorithm may generate errors because it is difficult to know which lines should cross the document columns in advance. If there are overlaps between the table column spaces and the document column spaces, this method will segment a whole table into at least two parts. Although some table lines may block the document column space, it is difficult to identify the last line of the table.

Algorithm 1: Sorting sparse lines across the document columns

```
Input: All sparse lines $SL = \cup_{i=1}^{m} sl_i$ in a page
Input: the information of Document columns
       $CO = \cup_{k=1}^{p} co_k$. The sparse lines that are
       located in the column $co_k$ will be stored in the
       corresponding vector $v_k$
Input: a vector $V_{cc}$ to store the cross-column sparse
       lines. $V_{cc} = \emptyset$
1  begin
2      foreach $sl_i \in SL$ do
3          $x_{sl_i} \leftarrow$ the starting X-axis of $sl_i$;
4          $x'_{sl_i} \leftarrow$ the ending X-axis of $sl_i$;
5          while the next document column $co_k$ exists do
6              $x_{co_k} \leftarrow$ the starting X-axis of $co_k$;
7              $x'_{co_k} \leftarrow$ the ending X-axis of $co_k$;
8              if $(x_{co_k} \leqslant x_{sl_i} < x'_{sl_i} \leqslant x'_{co_k})$ then
9                  $v_k = v_k + sl_i$;
10                 break;
11         if $sl_i$ fits no document column then
12             $v_{cc} = v_{cc} + sl_i$;
13 end
```

4.2 Algorithm 2: Without considering the document column information

To deal with the above special cases, we propose another text sorting algorithm without considering the document column information at the beginning. The algorithm has four steps: 1) obtaining all the non-duplicated Y-axis values of the sparse lines within an area; 2) sorting these Y-axis values from the top to the bottom; 3) ordering all the sparse lines in the area according to the sorted Y-axis values; 4) exporting the sparse lines in the area orderly. For the *inside-table* text sequence error problem, the area is the table itself. For the *beyond-table text sequence error* problem, this algorithm works better for the wide tables because the area refers to the whole page. All the sparse lines of cross-column tables will be exported out uninterruptedly. The high-risk case of this algorithm is the parallel tables. Parallel tables mean two tables that are located within two adjacent document columns and both tables have overlap Y-axis areas (see Figure 2a). Such cases happen very infrequently. In addition, they can be fixed easily by considering the width and the location of table captions.

4.3 Detecting the Table Boundary based on the sorted sparse lines

With the sorted sparse lines, we detect the table boundary by combining the table caption information. We define a keyword list, which lists all the possible starting keywords

Algorithm 2: Sorting sparse lines within a document column

```
Input: V, $V_{cc}, V_y, V_{sorted}$
1  foreach $v_k \in CO_k$ do
2      $V_y \leftarrow$ all the Y-axis values in $v_k$;
3      $V_y \leftarrow$ all the non-duplicated objects in $V_y$;
4      $V_y = V_y - V_{cc}$;
5      sorted($V_y$) ascendingly;
6      foreach $Y_q \in V_y$ do
7          foreach $sl_i \in V_k$ do
8              if $Y_{sl_i}$ == $Y_q$ then
9                  $V_{sorted}$ += $sl_i$;
10                 $V_k$ -= $sl_i$;
11 sort lines in $V_{cc}$ according to Y-axis;
12 insert $V_{cc}$ into $V_{sorted}$ according to Y-axis;
```

of table captions, such as "Table, TABLE, Form, FORM," etc. Each table caption starts with a keyword. If more than one table is displayed together, the keyword is very useful to separate the tables from each other. At the beginning, we check all the lines (not only the sparse line) in a page. Once we detect a line starting with a keyword, we treat it as a table caption candidate. Then we check the subsequent sorted sparse lines and merge them into a sparse area, according to the vertical distance between the lines. Different table captions provide different hints and restraints on whether we include the cross-column sparse lines or not.

If a table caption starts in the second document column (e.g., the right table in Figure 2a), the table boundary can not span the first column. If a table caption satisfies one of the following conditions, it is a cross-column table: 1) the width of the table caption is larger than the document column width; 2) the starting X-axis value of the caption is on the right of the midpoint of the first document column. Once we confirm a table as a cross-column table, we treat each Y-axis value as a table row. Otherwise, every table row can only include the sparse lines within the current document column. Because we will zoom in and analyze the detected table boundary carefully in the later table structure decomposition phase, we treat *recall* more importantly than *precision* by accepting false positive table boundary contents.

If a table caption is shorter than the document column width, and its beginning position aligns with that of the first document column (see the table caption in Figure 1), it is difficult to judge whether this table a cross-column table or not. For such a case, we assume it as a single-column table initially, after figure out the document column where the caption belongs to, we check the same Y-axis area in the next document column. If this area is also a sparse area, we merge the new area into the table boundary and treat this table as a cross-column table. Once we find a new table caption in the new area, it indicates the existing of parallel tables.

5 Experiments and Results

Our experiments can be divided into two parts: the evaluation of the text sequence recovering and the table boundary detection. A five-user study is implemented. Each user checked the sequence of the resorted sparse lines in 20 selected PDF pages. Half of them have within-table sequence errors and the other half have beyond-table sequence errors. The evaluation metric is *pairwise accuracy*. If H_R is the sequence decided by human judgement and A_R is the sequence decided by the algorithms, the *pairwise accuracy* can be defined as the fraction of times that our algorithms and human judges agree on the sequence: *pairwise accuracy* = $\frac{|H_R \cap A_R|}{H_R}$. For algorithm 1, the correct sequence of human judge is column by column. For algorithm 2, humans treat each page as a single-column page. Comparing such "*golden standard*", the *pairwise accuracy* of both algorithms are 100%. Still using the Figure 3a as the example, with our algorithms, the 13 non-duplicated Y-axis values after the sorting step are: *625.47974, 643.07983, 651.9796, 660.68005, 669.5801, 681.4798, 690.27966, 699.0797, 707.7796, 716.3797, 725.1799, 734.0797, 744.37976*. Because the text sequence error always starts with the left columns first, we do not need to sort the text pieces with the same Y-axis values according to the X-axis coordinate. After the sorting, the new sequence of these sparse lines are: *"Content (% m/m)", "Proposed method", "Reference", "Peak Peak Flame value", "Sample Element height area AAS (%m/m)", "Low melt Ag 43.9 44.1 45.1 44.46", "0.5 2.4 1.1", "Cu 15.6 15.2 15.3 14-16", "2.7 1.8 2.5", "Sample No.318+ Ag 25.7 25.2 25.7 ≈25", "2.1 2.1 1.4", "Cu 34.4 33.7 34.8 ≈35", "2.3 1.3 2.5"*.

The evaluation metrics for the table boundary detection are precision P and recall R. Given the number of true table lines in our detected table boundary A, the number of overlooked true positive table lines B, and the number of misidentified true negative non-table lines C, $P=\frac{A}{A+C}$, and $R=\frac{A}{A+B}$. The performance of two algorithms are listed in Table 1. The reason for C are some non-table sparse lines surrounding the table boundary. They usually also belong to the table data.(e.g., the short lines of the table caption and footnote). For single-column tables, Algorithm 2 may have more C because of the possible sparse lines with similar Y-axis values in the next document column. Because the widths of cells in single-column table can not be large and the noise lines in table boundary can be easily removed in later table structure decomposition step, we prefer high recall values. For cross-column tables, algorithm 1 will cut each table into at least two parts. A merge postprocessing is required. Algorithm 2 achieves better performance on such tables. In summary, in the real application when we can not know the table type in advance, algorithm 2 works better than algorithm 1. For tables with within-table sequence errors, our recovering algorithms do not fulfill much performance improvement on both P and R because all the lines belong to the table boundary come together without interruption. However, for the tables with beyond-table sequence errors, the performance is much worse without our algorithms: with the same test PDFs, the R is only 63.5% without our algorithms because a table will be segmented into several parts and some of them are filtered out because of the small scope.

Table 1. The performance evaluation of two text sequence resorting algorithms

	Algo1(P)	Algo1(R)	Algo2(P)	Algo2(R)
Cross-column tables	95.7%	49.8%	100%	99.6%
Single-column tables	96%	94.8%	94.5%	99.8%

6 Conclusion

In this paper, we analyze a typical problem shared by the PDF text extraction tools: the text sequence error. We propose two algorithms to recover the sequence of extracted sparse lines, which improve the table content collection. The experimental results not only compare the performance of both algorithms, but also demonstrate the effectiveness of text sequence recovering for the table boundary detection. The results show that the second algorithm achieves better results for both single-column table and cross-column table. It is proved that the text sequence recovering work plays a crucial role in the table boundary detection field.

References

[1] R. Cattoni, T. Coianiz, S. Messelodi, and C. Modena. Geometric layout analyis techniques for document image understanding: a review. *ITC-IRST Technical Report 9703-09.*

[2] H. Chao and J. Fan. Layout and content extraction for pdf documents. In *DAS04*, pages 213–224, 2004.

[3] Y. Liu, P. Mitra, and C. L. Giles. A fast preprocessing method for table boundary detection: Narrowing down the sparse lines using solely coordinate information. In *DAS*, 2008.

[4] Y. Liu, P. Mitra, and C. L. Giles. Identifying table boundaries in digital documents via sparse line detection. In *CIKM*, pages 1311–1320, 2008.

[5] X. Wang. Tabular abstraction, editing, and formatting. In *Ph.D. Thesis, Dept. of Computer Science, University of Waterloo*, 1996.

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

[7] B. Yildiz, K. Kaiser, and S. Miksch. pdf2table: A method to extract table information from pdf files. In *Proceedings of the 2nd Indian International Conference on Artificial Intelligence (IICAI05), Pune, India, 2005.*