pdf2table: A Method to Extract Table Information from PDF Files

Burcu Yildiz, Katharina Kaiser, and Silvia Miksch

Institute of Software Technology & Interactive Systems
Vienna University of Technology, Vienna, Austria
{yildiz, kaiser, silvia}@asgaard.tuwien.ac.at

Abstract. Tables are a common structuring element in many documents, such as PDF files. To reuse such tables, appropriate methods need to be develop, which capture the structure and the content information. We have developed several heuristics which together recognize and decompose tables in PDF files and store the extracted data in a structured data format (XML) for easier reuse. Additionally, we implemented a prototype, which gives the user the ability of making adjustments on the extracted data. Our work shows that purely heuristic-based approaches can achieve good results, especially for lucid tables.

1 Introduction

The amount of accessible data we are facing today makes it necessary to develop efficient Information Engineering concepts and tools to better process and use the data. Information Engineering comprises such a wide dimension of sub-areas that one cannot expect that a single concept or tool can fit the needs of all [1]. One of these sub-areas is the field of Information Extraction (IE). IE is the task of extracting relevant facts from text and representing them in some useful form. The development of this field was influenced and fostered by a series of Message Understanding Conferences (MUCs) starting in 1987 which served as a platform for evaluating different IE projects developed by different sites [2,3]. The field of IE can also be split into some sub-tasks.

One sub-task is the task of Table Extraction (TE) which is the subject of this paper. This task is important, because tables are among the most common means of presenting and structuring data with a high information density. However, it is not an easy task, because tables can be of varying formats. For example, some tables could have lines in order to point out the cell boundaries, whereas others could have only white spaces to achieve a table view. The only thing each table will certainly have is content.

Further, we concentrated only on PDF files as input files. This data format is widely known and used, because it allows users to create files that look the same on different output devices, no matter in which environment they were created.

Extracting different kinds of data and information from whole PDF files is a field of research itself. Various tools were developed to support the extraction process. A comparison [4] showed that the tool with the most useful output for our purpose was the *pdftohtml*[1] tool developed by Gueorgui Ovtcharov and Rainer Dorsch. This tool

[1] http://pdftohtml.sourceforge.net

returns all text elements (i.e., strings) in a PDF-file with their absolute coordinates in the original file. Using this tool, our task became to extract table information from semi-structured text files utilizing their absolute coordinates.

2 Table Extraction

Table Extraction (TE) is the task of detecting and decomposing table information in a document. This task attracted the attention of researchers because tables are one of the most used elements to present and structure data and they should be extracted for reuse.

While human beings can easily recognize and understand tables, things are different for computers, because tables do not have any identifying characteristic in common. They can contain delimiters ranging from graphical boundary lines to point out the boundaries and the separation between rows and cells, to only white spaces to achieve a table view. Further, they can vary in terms of containing spanning rows and/or columns. Another point that makes TE harder is that tables can contain different types of content, such as text, figures, mathematical formulas, etc. [5] We had to take all the explained difficulties into account in developing our approach.

Our work is based on the data returned by the *pdftohtml* tool (refer Section 1). For each text chunk in the PDF file it returns a text element in XML with the following attributes:

- top = vertical distance from the top of the page
- left = horizontal distance from the left border of the page
- width = width of the text chunk
- height = height of the text chunk
- font = this attribute describes the size, family, and color of the text chunk

We restricted our work to utilize only these five attributes for extracting table information and not, for example, graphical components like lines, etc. After applying the *pdftohtml* tool, we had to extract table information from an XML document with text elements describing the absolute position of a text chunk in a PDF file.

We have explored different kinds of tables according to their structure to develop several heuristics. These heuristics can be grouped in two main categories: (1) heuristics intended to recognize a table and (2) heuristics intended to decompose a table.

Table Recognition. This task deals with the problem of identifying a ”construct” as a table. The level of difficulty of this task depends, among others, on the document in which the table is embedded. As we deal with an XML document which does not mark-up tables, we have to identify a portion of text elements as a table only by means of the knowledge of the absolute coordinates of the text elements.

Table Decomposition. After detecting a part of a file as a table, the next step is to decompose the table as close to the original as possible. This task includes the correct identification of header elements, their spanning behavior (i.e., how many columns or rows are spanned), the correct assigning of data cells to header elements, and so on.

3 Our Approach

Our approach is based on heuristics, which we derived from comparing different kinds of tables according their composition. We grouped our heuristics in tasks of table recognition and table decomposition. First, we explain our preprocessing and then the heuristics. All the algorithms are listed with a basic explanation. Afterwards, we give a coherent example, which illustrates all the different steps.

To ease the understanding of our heuristics, we define some basic terms, which will be used throughout the paper.

- Text: contains a string and five attributes (top, left, width, height, font)
- Line: contains text objects which are assumed to be on the same line in the original file
- Single-Line: line object with only one text object
- Multi-Line: line object with more than one text object
- Multi-Line Block: set of continuous multi-line objects

Basically, we assume the input document as a single column document. By using a user interface the user can actually tell the implemented prototype the number of columns of the document to achieve better results.

3.1 Preprocessing

The *pdftohtml* tool returns text chunks and their absolute coordinates in the PDF file in the same order as they were inserted into the original file. Because each author can insert the text in the order she/he wants, you cannot rely only on the ordering of the text elements to make decisions. To avoid such uncertainties we first sort all text elements according to their top values.

Ascii	Sign	Ascii	Sign	Ascii	Sign
048	0	050	2	052	4
049	1	051	3	053	5

Fig. 1. Example of a table in a PDF file

The original ordering of the text elements in Fig. 1 can have several forms. One possible ordering could be: Ascii, Sign, Ascii, Sign, Ascii, Sign, 048, 0, 050, 2, and so on. Depending on the author, another ordering could be: Ascii, 048, 049, Sign, 0, 1, and so on. If we sort all the text elements with respect to their top-values we can be sure that we always get the same ordering, no matter how the author has inserted the text chunks. For Fig. 1 the sorted ordering is: Sign, Sign, Sign, Ascii, Ascii, Ascii, 048, 0, 050, and so on.

After this sorting process we want to assign text objects that are on the same line to a line object. Our heuristic for this task is described in Algorithm 1.

Algorithm 1. *Merge text elements on the same line to line objects*

```
for each Text t {
    Line pl = last Line in the Line list
    if (t.top or t.bottom lies between pl.top and pl.bottom) {
        add t to pl;
        actualize values of pl.top and pl.bottom;
    } else {
        create new Line and add t to the new Line;
        set top and bottom values of the new Line;
        add new Line to the Line list;
    }
}
```

After applying Algorithm 1, we have all the lines in the PDF file in our line object list. We can start with the table recognition task.

3.2 Table Recognition

In this task, we utilize the gained information from our pre-processing to identify the tables in the document. Our basic assumption for recognizing tables is: ”Tables must have more than one column”. This indicates that each multi-line object can be a data row of a table and each multi-line block object can actually be a table. Based on these assumptions we describe our table recognition heuristic in Algorithm 2.

Algorithm 2. *Classify single-line and multi-line objects and detect multi-line block objects*

```
multi-modus = false;
for each Line line {
    if (number of Text objects in line > 1) {
        mark line as Multi-Line;
        if (multi-modus == false) {
            create new Multi-Line Block;
            add new Multi-Line Block to Multi-Line Block list;
            multi-modus = true;
        } else {
            Multi-Line Block mlb = last added Multi-Line Block
                to the Multi-Line Block list;
            add line to Lines in mlb;
        }
    } else
        if (number of Text objects in line == 1) {
            Text t = the Text in line;
            Multi-Line Block mlb = last added Multi-Line Block
                to the Multi-Line Block list;
            if (t belongs to mlb)
                add line to Line in mlb;
           else  // single-line
                multi-modus = false;
        }
}
```

After this first classification of line objects as single-line and multi-line objects and detecting multi-line block objects we have generated a heuristic to merge multi-line block objects that may belong to the same table. We selected a threshold value of five and assume that if there are more than five single-line objects between two multi-line block objects, than these multi-line block objects represent two distinct tables. Algorithm 3 presents this heuristic.

Algorithm 3. *Merge multi-line block objects which may belong to the same table*

```
for each Multi-Line Block mlb {
    p_mlb = previous Multi-Line Block in the Multi-Line Block list;
    n_mlb = next Multi-Line Block in the Multi-Line Block list;
    for each Line between mlb and p_mlb
        try to merge Line to mlb;
    if (number of Lines between mlb and p_mlb <=5 and mlb and
        p_mlb on the same page) {
        merge mlb and p_mlb;
    }
    for each Line between mlb and n_mlb
        try to merge Line to mlb;
    if (Line between mlb and n_mlb <=5 and mlb and
        n_mlb on the same page) {
        merge mlb and n_mlb;
    }
}
```

After the merging process we have all multi-line block objects that can be a table. We are not taking multi-line block objects with less than two rows into account for further processing. The next step is to decompose the remaining multi-line block objects to tables.

3.3 Table Decomposition

Having all possible tables detected we can concentrate on decomposing the information in these multi-line block objects. Due to the fact that each text element will be in (at least) one column, we simplify our decomposition task and try to find the appropriate column for each text element in a multi-line block object (refer Algorithm 5). We start with no column at hand. For the first text element there will be a new column created. After that, for each text element the boundaries of the text element will be compared with the existing columns' boundaries. According to that, if a text elements' horizontal boundaries fit into a columns' boundaries it will be added to this column, if not, a new column will be created and the text element will be added into it. If more than one text element in a line falls into the boundaries of a column the spanning will be increased. After adding a text element to a column, the boundary values of the column are actualized.

Our heuristic for this task is described in Algorithm 4.

Algorithm 4. *Decompose the columns of each multi-line block object*

```
for each Multi-Line Block mlb {
    for each Line line in mlb {
        for each Text t in line {
            find appropriate column for t;
            if (found)
                add t to cells in column;
            else {
                create new column;
                add column at the appropriate position in the column list;
            }
        }
    }
    create table with all these columns;
}
```

Finding the appropriate column for a text object requires a heuristic itself, which is described by Algorithm 5.

Algorithm 5. *Assign text element to a column*

```
input: Text t
for each column c {
    if (t overlaps c or c overlaps t)
        return c;
    else
        if (t in c or c in t)
            return c;
        else
            return null;
}
```

After all these processes we have a list of columns which together compose the tables. Now, the only thing to do is to merge cells with the same content to one cell with a greater spanning value.

3.4 An Example

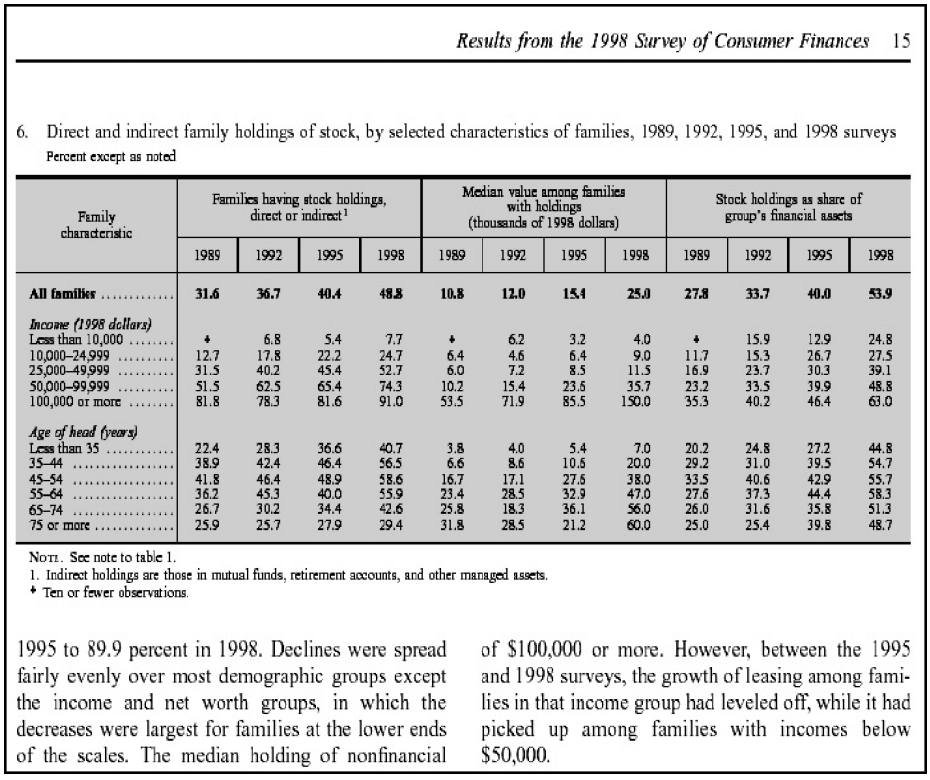
In the following we will give an example to illustrate several steps of our approach.

Assume that we have as input the PDF file with a page like in Fig. 2. Of course, the PDF file contains not only the table but also text paragraphs, footnotes, etc., too.

After getting the results from the *pdftohtml* tool we can go on with our approach. Our first step is to sort all the text elements in respect to their top attributes. Assume that we have already identified the text elements before the table and let us begin with the text elements in the table (refer Fig. 3).

In Fig. 3, after the sorting process we have the following ordering:

”Median value among families”, ”Families having stock holdings”, ”with holdings”, ”direct or indirect”, ”Family”, ”(thousands of 1998 dollars)”, ”characteristic”, ”1989”, ”1992”, ”1995”, ”1998”, ”1989”, and so on.



6. Direct and indirect family holdings of stock, by selected characteristics of families, 1989, 1992, 1995, and 1998 surveys

Percent except as noted

Family characteristic	Families having stock holdings, direct or indirect[1]				Median value among families with holdings (thousands of 1998 dollars)				Stock holdings as share of group's financial assets			
	1989	1992	1995	1998	1989	1992	1995	1998	1989	1992	1995	1998
All families	**31.6**	**36.7**	**40.4**	**48.8**	**10.8**	**12.0**	**15.4**	**25.0**	**27.8**	**33.7**	**40.0**	**53.9**
Income (1998 dollars)												
Less than 10,000	*	6.8	5.4	7.7	*	6.2	3.2	4.0	*	15.9	12.9	24.8
10,000–24,999	12.7	17.8	22.2	24.7	6.4	4.6	6.4	9.0	11.7	15.3	26.7	27.5
25,000–49,999	31.5	40.2	45.4	52.7	6.0	7.2	8.5	11.5	16.9	23.7	30.3	39.1
50,000–99,999	51.5	62.5	65.4	74.3	10.2	15.4	23.6	35.7	23.2	33.5	39.9	48.8
100,000 or more	81.8	78.3	81.6	91.0	53.5	71.9	85.5	150.0	35.3	40.2	46.4	63.0
Age of head (years)												
Less than 35	22.4	28.3	36.6	40.7	3.8	4.0	5.4	7.0	20.2	24.8	27.2	44.8
35–44	38.9	42.4	46.4	56.5	6.6	8.6	10.6	20.0	29.2	31.0	39.5	54.7
45–54	41.8	46.4	48.9	58.6	16.7	17.1	27.6	38.0	33.5	40.6	42.9	55.7
55–64	36.2	45.3	40.0	55.9	23.4	28.5	32.9	47.0	27.6	37.3	44.4	58.3
65–74	26.7	30.2	34.4	42.6	25.8	18.3	36.1	56.0	26.0	31.6	35.8	51.3
75 or more	25.9	25.7	27.9	29.4	31.8	28.5	21.2	60.0	25.0	25.4	39.8	48.7

NOTE. See note to table 1.

1. Indirect holdings are those in mutual funds, retirement accounts, and other managed assets.

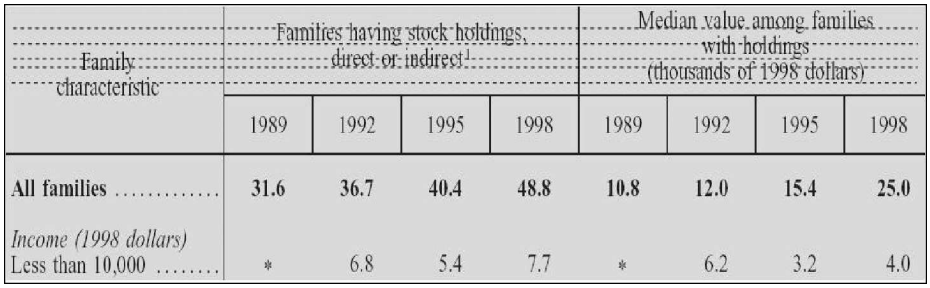
* Ten or fewer observations.

1995 to 89.9 percent in 1998. Declines were spread fairly evenly over most demographic groups except the income and net worth groups, in which the decreases were largest for families at the lower ends of the scales. The median holding of nonfinancial

of $100,000 or more. However, between the 1995 and 1998 surveys, the growth of leasing among families in that income group had leveled off, while it had picked up among families with incomes below $50,000.

Fig. 2. Example of a complex table in a PDF file

Now, Algorithm 1 is applied to create the line objects. Based on the ordering the first text element that is saved in a line object is ”Median value among families”. Thus, a new line object is created and the top and bottom values are actualized in respect to the added text element. The next one is ”Families having stock holdings” and we must look whether we can put this text in an existing line object or not. The first dashed line (see Fig. 3) marks the bottom of the line object we just created. As you can see the current text elements’ top value is between the top and the bottom value of our first line object and thus can be added to this line. After adding, the line objects’ top and bottom values are actualized. This procedure is applied until all text objects have been found in a line object (the last text object in our example is ”1989”). The text elements in this line object are still sorted according to their top values. This ordering is of no use anymore, because we want to gain the text chunks that semantically belong together. For example, we want ”Family” and ”characteristic” merged. Thus, we next sort the text elements in the line object according to their left values. After that we have the wanted ordering, thus ”Family characteristic”, ”Families having stock holdings direct or indirect”, and so on (refer Fig. 3).



Family characteristic	Families having stock holdings, direct or indirect[1]				Median value among families with holdings (thousands of 1998 dollars)			
	1989	1992	1995	1998	1989	1992	1995	1998
All families	**31.6**	**36.7**	**40.4**	**48.8**	**10.8**	**12.0**	**15.4**	**25.0**
Income (1998 dollars)								
Less than 10,000	*	6.8	5.4	7.7	*	6.2	3.2	4.0

Fig. 3. Illustrating the ordering in which the text elements are added into line objects

After building all line objects in this page we have to classify all line objects as single-line object or multi-line object. Algorithm 2 marks successive multi-line objects as multi-line block objects. Because we have no other multi-line block object in this example we do not have to merge anything (refer Algorithm 3).

The next step is to create columns and assign the text objects to their corresponding columns. This step is done by Algorithm 4 and Algorithm 5. For our first text object in the first line object (”Family characteristic”) we have to build a new column. For all the text objects in the line objects we have to look whether there exists a column to which that text object can be assigned. If so, we simply add the text object to this column. If not, we create a new column and add this text object to the new one. In both cases, we actualize the columns’ horizontal boundaries according to the new added text element.

A text object can be assigned to a column if one of the following four possibilities appears (refer Fig. 4):

1. The text is positioned completely within the horizontal boundaries of the column
2. Left border of the text is positioned in the horizontal column boundaries
3. Right border of the text is positioned in the horizontal boundaries of the column
4. The text spans the horizontal boundaries of the column

After this procedure we have a table consisting of more than one column. For the first five columns of our example in Fig. 2 we get the resulting columns presented in Table. 1.

Finally, we have to identify neighbor cells with the same content and merge them. In our case the four cells with the content ”Families having stock holding direct or indirect” are merged into one single cell with a column spanning of four. These are the main steps of our approach to extract table information from PDF files.

3.5 Limitations

Because of the complexity of the task and the used heuristics, which cannot cover all possible table structures, one cannot assume that the approach always returns correct results. For example, our approach cannot distinguish between hidden tables (i.e., tables that are not labeled as such in the original file) and real tables. Further, tables that

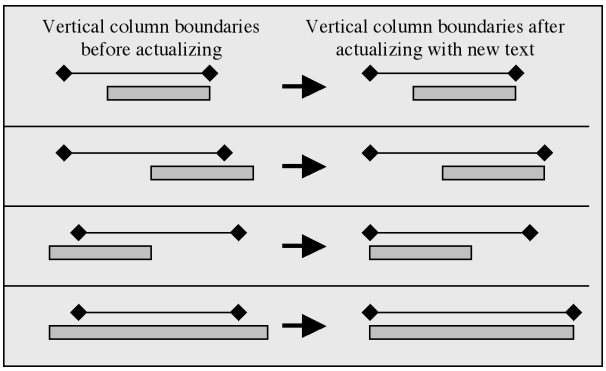


Fig. 4. The four possibilities for assigning a text object to a column

Table 1. Columns after applying Algorithm 4.

Family characteristic	Families having stock holding direct or indirect	Families having stock holding direct or indirect	Families having stock holding direct or indirect	Families having stock holding direct or indirect
Null	1989	1992	1995	1998
All Families	31.6	36.7	40.4	48.8
...				
...				

are positioned vertically on a page cannot be captured. There are also several possible errors, for example, text chunks that do not belong together are merged, multi-line block objects that belong together are not merged, data cells are assigned to wrong columns, and so forth. It is also possible that areas that are not tables are identified as such. This is the case, for example, with bulleted lists, etc.

To overcome these limitations we also implemented a graphical user interface which gives the user the ability of making adjustments on the extracted data. The user can make adjustments on cell level (e.g. delete cells, merge cells, edit content of cell, etc.) or on table level (e.g. delete table, merge tables, delete/insert rows or columns).

The main limitation of the tool is that it is based on the results of the *pdftohtml* tool. If this tool returns wrong information or no information at all, our approach cannot be applied. For example, PDF files sometimes contain only the image of a table and not text chunks which are inserted by an author. In such a case, the *pdftohtml* returns no useful information. We stated this limitation as the main limitation, because the user cannot do anything about it. The graphical user interface will not help, either.

3.6 Evaluation

The evaluation of an Information Extraction System is a non-trivial issue. Therefore, we can say that the MUCs' scoring program represented an important first step in the right direction [6]. These conferences served as an evaluation platform where systems from different sites were evaluated by using different measures. Over the years the recall and precision measures established themselves and are widely accepted as a means for giving evidence about a system's performance. Currently, some research goes in the direction of finding new and proper measures for evaluating table-processing approaches [7].

However, it is hard to predict how good a measure reflects the real situation for a current approach. Our approach, for example, consists of several iterative steps and a failure in the first step would affect the end result to an unpredictable extent. But it would be very hard to evaluate the performance of each heuristic separately. Thus, we decided to evaluate the end result using the mentioned most established measures in the IE community, namely the recall and precision measures [2].

We evaluated the table recognition and decomposition task separately and transformed the formula for recall and precision according to the tasks. The formula for the table recognition task is as follows:

$$\textit{Recall} = \frac{number\ of\ correctly\ identified\ tables}{number\ of\ tables\ in\ the\ document}$$

$$\textit{Precision} = \frac{number\ of\ correctly\ identified\ tables}{number\ of\ identified\ tables}$$

The formula for recall and precision for the table recognition task is as follows:

$$\textit{Recall} = \frac{number\ of\ correctly\ assigned\ data\ cells}{number\ of\ data\ cells\ in\ the\ original\ table}$$

$$\textit{Precision} = \frac{number\ of\ correctly\ assigned\ data\ cells}{number\ of\ extracted\ data\ cells}$$

To determine how well our approach performs on the task of table recognition and decomposition we implemented our approach and evaluated it with several PDF files containing various tables. For that purpose, we used two test corpora. One consists of documents with lucid (or unsophisticated) tables and the other consists of documents with complex tables. The documents do not only contain tables, but also text paragraphs, graphics, etc.

Different application fields are using different kinds of tables. But within a particular field the structure of the tables are quite similar. If we would test our approach on such a data set, the evaluation results would not reflect the real performance of our approach. To overcome this problem, we decided to generate our test corpora randomly with PDF files available on the World Wide Web. Most of the PDF files contain only one table and they came from various fields like research, sports, statistics, etc. They are also not restricted to be from a specific domain or language.

Tables 2 and 3 show the evaluation results of our approach for the table recognition and table decomposition tasks.

The run-time of the heuristics increase with the number of pages in a PDF file. In order to achieve faster results the user should start the tool for only the pages of interest.

Table 2. Evaluation results of the table recognition task

	Amount of samples	Recall	Precision
Lucid tables	50	0.84	0.97
Complex tables	100	0.92	0.95

Table 3. Evaluation results of the table decomposition task

	Amount of samples	Recall	Precision
Lucid tables	50	0.88	0.97
Complex tables	100	0.81	0.83

4 Related Work

The Information Engineering community has realized the importance of TE earlier. Some approaches handle images of documents as input and therefore merely focus on layout components (table boundaries, cell boundaries, etc.) to detect and decompose tables. Other approaches, on the other hand, have to deal with unstructured or semi-structured text where only little or no information about the layout is given. In such cases, the developers try to extract the tables by using parameters like spacing, etc.

We can categorize the existing approaches in three main categories [8]:

- **Predefined layout-based approach:** In this approach there exist several templates for possible table structures. The input documents are scanned and portions that fit a template are identified as tables. The limitation is that one cannot define so many templates so that all possible table structures are covered with them.
- **Heuristic-based approach:** This approach makes use of a set of rules, which are created to make decisions.
- **Statistical or optimization-based approach:** This approach uses statistical measures obtained by offline training. The estimated parameters are then used for decision-making.

Tupaj and colleagues [9] focus on examining a document image and its content. In their algorithm, first the document is segmented and analyzed by means of image processing techniques in order to detect potential table areas. These parts are then passed

to an OCR engine that extracts the text. Finally, the extracted text is analyzed and table components are isolated. Tupaj and colleagues used two kinds of tables for the testing process (technical tables, financial tables). They show that their approach gains different results for the two kinds of table classes.

Ng and colleagues [10] apply Machine Learning techniques to gain a domain independent and reliable system. They developed an approach that handles plain text in ASCII characters as input. Their learning method uses purely surface features like the relative locations of characters in a line and across lines, etc. They split their task into three sub-tasks: recognizing table boundary, column, and row, in that order. Note that they only focus on detecting tables, columns, and rows and not on the content.

Ramel and colleagues [5] developed a method for detection and extraction of the graphic lines. To deal also with tables with no or little graphic marks they utilize the regularities of the text elements alone. They describe their approach as a top-down study of regularities, whereas first the global configuration of the blocks of text are examined, then more local characteristics (alignment, spacing) are taken into account, and finally the lines of text are studied.

Pinto and colleagues [11] presented a model of table extraction that utilizes both content and layout of tables by using conditional random fields (CRFs). CRFs are undirected graph models that can be used to calculate conditional probability of values on designated input nodes. Pinto and colleagues first label each line of a document with a tag (non-extraction label, header label, data row label, caption label). Then they describe features used by a heuristic table extractor. They used CRFs because CRFs support the use of many rich and overlapping layout and language features. After a training phase they tested their dataset and achieved good results. Though, the current state of their model can only locate labels and tag lines but knows nothing about the columns and cannot distinguish between data cells and header cells.

Although there is some existing work about extraction from PDF files, there is no special focus on the extraction of tables. The full version of Adobe Acrobat [2] has a table extraction feature but it can only extract lucid tables correctly and this only after the user marks the table. The other existing approaches were not applicable for our purposes, because they use to some extent graphical delimiters for the extraction. The used *pdftohtml* tool returns only text elements with some attributes and no graphical components at all. Further, we wanted to preserve generality in the sense of not being limited to a specific domain, language, or a set of table models etc. Thus, we decided to develop a heuristic-based approach that works only with the few attributes of the tool and nothing else.

5 Conclusion

This paper has presented a method for extracting table information by utilizing only the absolute position of text elements in a file, concretely in PDF files. Further, a prototype was generated in order to evaluate the performance of the method. Experiments on several PDF documents with 150 tables demonstrated our results.

[2] http://www.adobe.com

Based on our evaluation results we can say that our approach performs very well on the table recognition task for lucid and complex tables. Possible errors are an unintended merge of two adjacent tables or erroneous table splits.

Regarding the table decomposition task we can say that our approach works very well on lucid tables. Anyhow, the performance and recall values on complex tables are also fairly good. The difference between these results came mainly from the appearance of multi-line cells in complex tables. To decide whether two cells have to be merged or not requires some amount of natural language understanding and our approach does not handle any language-specific features. Therefore, we can only make decisions based on the distance between two cells, which is not always reliable. It is possible that two cells that have to be merged are not merged by the approach or that two adjacent cells are falsely merged.

Our approach to the extraction of tables in PDF files, is domain and language independent. Future work may involve using more statistical techniques for utilizing regularities in tables in order to achieve better results. Such techniques could, for example, help to avoid false-positives in both of the tasks.

References

1. Miller, R.L.: Information engineering: A balanced approach to information systems requirements analysis and design. In: Proceedings of the IEEE National Aerospace and Electronics Conference. (1995) 672–679
2. Appelt, D.E.: Introduction to information extraction. AI Communications **12** (1999) 161–172
3. Riloff, E.: Information extraction as a stepping stone toward story understanding. In Ram, A., Moorman, K., eds.: Understanding Language Understanding: Computational Models of Reading. MIT Press (1999)
4. Yildiz, B.: Information extraction – utilizing table patterns. Master's thesis, Vienna University of Technology (2004)
5. Ramel, J.Y., Crucianu, M., Vincent, N., Faure, C.: Detection, extraction and representation of tables. In: Proceedings of the Seventh International Conference on Document Analysis and Recognition, Washington DC., IEEE Computer Society (2003) 374–378
6. Lehnert, W., Cardie, C., Fisher, D., McCarthy, J., Riloff, E., Soderland, S.: Evaluating an Information Extraction system. Journal of Integrated Computer-Aided Engineering **1** (1994)
7. Hu, J., Kashi, R., D., L., G., W.: Evaluating the performance of table processing algorithms. International Journal on Document Analysis and Recognition **4** (2002)
8. Wang, Y.: Document Analysis: Table Structure Understanding and Zone Content Classification. PhD thesis, Washington University (2002)
9. Tupaj, S., Shi, Z., Chang, C., Alam, H.: Extracting tabular information from text files. Eecs, Tufts University, Medford (1996)
10. Ng, H., Lim, C., Koo, J.: Learning to recognize tables in free text. In: Proceedings of the 37th Conference on Association for Computational Linguistics. (1999) 443–450
11. Pinto, D., McCallum, A., Wei, X., Bruce, W.: Table extraction using conditional random fields. In: Proceedings of the 26th ACM SIGIR. (2003)