TabbyPDF: Web-Based System for PDF Table Extraction

Alexey Shigarov[1,2], Andrey Altaev[1], Andrey Mikhailov[1], Viacheslav Paramonov[1,2], and Evgeniy Cherkashin[1,2]

[1] Matrosov Institute for System Dynamics and Control Theory of SB RAS, 134 Lermontov st., Irkutsk, Russia,
[2] Institute of Mathematics, Economics and Informatics, Irkutsk State University, 20 Gagarin blvd., Irkutsk, Russia
shigarov@icc.ru,
WWW home page: http://cells.icc.ru

Abstract. PDF is one of the most widespread ways to represent non-editable documents. Many of PDF documents are machine-readable but remain untagged. They have no tags for identifying layout items such as paragraphs, columns, or tables. One of the important challenges with these documents is how to extract tabular data from them. The paper presents a novel web-based system for extracting tables located in untagged PDF documents with a complex layout, for recovering their cell structures, and for exporting them into a tagged form (e. g. in CSV or HTML format). The system uses a heuristic-based approach to table detection and structure recognition. It mainly relies on recovering a human reading order of text, including document paragraphs and table cells. A prototype of the system was evaluated, using the methodology and dataset of "ICDAR 2013 Table Competition". The standard metric F-score is 93.64% for the structure recognition phase and 83.18% for the table extraction with automatic table detection. The results are comparable with the state-of-the-art academic solutions.

Keywords: table understanding, table extraction, table detection, table recognition, document analysis, PDF accessibility

1 Introduction

PDF is one of the most widespread ways to represent non-editable documents. P. Ydens guesstimates in his keynote address[3] on PDF Technical Conference 2015 that there might be 2.5 trillion PDF files created each year. Many of PDF documents are machine-readable but remain untagged. They have no tags for identifying layout items, including running titles, sections, paragraphs, figures, lists, and tables. A recent estimation reported by Nganji [12] claims that 95.5% of all scientific articles published by four leading publishers are untagged PDF documents. One of the important challenges with these documents is how to

[3] https://www.pdfa.org/pdf-in-2016-broader-deeper-richer

extract tabular data from them. PDF tabular data may be of interest in various data extraction applications, including financial analytics (e. g. [1]), knowledge base construction (e. g. [7]), augmentation of Open Data [2] and Linked Open Data resources [19].

Table extraction as a part of *table understanding* [6] includes two phases: *table detection*, i. e. recovering the bounding box of a table in a document, and *table structure recognition*, i. e. recovering its rows, columns, and cells.

Many table extraction methods traditionally deal with only document images or plain-text [3, 26]. Usually, they can also be applied to machine-readable PDF documents. However, the PDF-to-image conversion leads to the loss of valuable information, including text chunks and positions, font features, an order of appearance PDF instruction in a file, vector ruling lines, and positions of a drawing cursor. In comparison with images and plain-text, PDF is a richer representation of documents. PDF documents can contain machine-readable text as well as vector ruling lines. We expect that extracting tables from machine-readable PDF documents directly can provide more accurate results.

Several methods and tools for PDF table extraction are proposed in two last decades. Some of them are discussed in the recent surveys [2, 3, 9, 10]. Ramel et al. [16] consider two techniques for detecting and recognizing tables from documents in an exchange format like PDF. The first is based on the analysis of ruling lines. The second is to analyze the arrangement of text components. Hassan et al. [8] expand these ideas for PDF table extraction. In the project TableSeer, Liu et al. [11] propose methods for detecting tables in PDF documents and extracting metadata (headers). They use text arrangement, fonts, whitespace, and keywords (e. g. "Table", "Figure"). Oro et al. [14] present PDF-TREX, a heuristic method where PDF table extraction is realized as building from content elements to tables in a bottom-up way.

Yildiz et al. [27] propose a heuristic method for PDF table extraction using pdftohtml[4] for generating its input. They also use the pdftohtml tool to prepare their input. However, this tool occasionally makes mistakes in combining text chunks, which are located too close to each other, thus the input can be corrupted. Nurminen [13] in his thesis describes comprehensive PDF table detection and structure recognition algorithms that have demonstrated high recall and precision on "ICDAR 2013 Table Competition" [4]. Some of them are implemented in Tabula[5], a tool for extracting tabular data from PDF. Rastan et al. [17] consider a framework for the end-to-end table processing including the task of table structure recognition. Moreover, Rastan et al. [18] suggest using an ad-hoc document analysis leading to better table extraction. Their wrapper able to detect features such as page columns, bullets, and numbering. Perez-Arriaga et al. [15] combines layout heuristics with a supervised machine learning method based on k-nearest neighbors to extract tables from untagged PDF documents. TAO, their system, promises to be an efficient, comprehensive and robust solu-

[4] `http://pdftohtml.sourceforge.net`
[5] `http://tabula.technology`

tion for both stages: table detection and cell structure recognition, that does not depend on fixed patterns or layouts of tables or documents.

We develop TabbyPDF, a novel web-based system for PDF table extraction from machine-readable untagged documents. This extends our previous work for table structure recognition [23]. The system exploits a set of customizable ad-hoc heuristics for table detection and cell structure reconstruction based on features of text and ruling lines presented in PDF documents. Most of them such as horizontal and vertical distances, fonts, and rulings are well known and used in the existing methods. Additionally, we propose to exploit the feature of appearance of text printing instruction in PDF files and positions of a drawing cursor. We also demonstrate experimental results based on the existing competition dataset, "ICDAR 2013 Table Competition". The standard metric F-score is 93.64% for the structure recognition phase and 83.18% for the table extraction with automatic table detection. The results are comparable with the state-of-the-art academic solutions.

2 PDF Table Extraction

The process of PDF table extraction involves the following phases:

1. *data preparation*, to recover text blocks presented words and ruling lines from instructions of a source PDF document;
2. *text line and paragraph extraction*, to recover text blocks presented lines and paragraphs;
3. *table detection*, to recover a bounding box of each table located on a page;
4. *table structure recognition*, to recover a cell structure of a detected table.

2.1 Data Preparation

We operate two kinds of objects:

- *text blocks* represent the text items: words, lines, paragraphs (including cell textual content);
- *ruling lines* serve as a representation of the graphic items: separator lines, and borders (including cell borders).

Text block consists of the following data:

- `bbox` — bonding box with four coordinates: `xl` — left, `yt` — top, `xr` — right, and `yb` — bottom, the x-coordinate increases from left to right, and y-coordinate increases from top to bottom;
- `font` — font with the attributes: `ff` — family (string value), `fs` — size in points, `fb` — bold or not, `fi` — italic or not;
- `order` — order of the appearance of text chunks in the source PDF file: `start` — index of a start text chunk in the order, and `end` — index of an end text chunk in the order.
- `ws` — width of a regular space in this font.

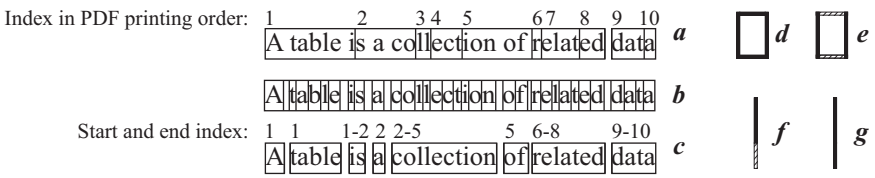

Fig. 1. Data gathered from an untagged PDF document: transforming text chunks (a) and positions (b) into text blocks representing words (c); splitting rectangles (d) into segments (f); merging segments (e) into ruling lines (g).

On this stage, each text block presented as a word is extracted from text chunks and positions of a PDF document, as shown in Fig. 1, a–c. We split all PDF text chunks (Fig. 1, a) into text positions (characters) (Fig. 1, b) and merge them into text blocks (words) with removing space characters and inducing the order of their appearance (Fig. 1, c).

Ruling line is characterized by four coordinates (`xl` — left, `yt` — top, `xr` — right, and `yb` — bottom) and `visibility` (visual or not). We separately recover both visual ruling lines presented borders and invisible ones that can usually represent traces of cursor motion (`moveto`, `lineto`). A ruling line can be originally presented by several PDF instructions for printing lines and rectangles. We split rectangles into segments by their bounds (Fig. 1, d–e). All segments are merged into ruling lines (Fig. 1, f–g).

2.2 Text Line and Paragraph Extraction

This stage aims to combine text blocks in order that each text block presents a textual content of one paragraph or one cell. We exclude preliminarily text blocks containing only itemization symbols (e. g. bullet, square). Typically, itemization symbols are visually detached from the rest of text chunks by long spaces. This lead to improperly recovered columns. We try to prevent these errors, eliminating these symbols from the further process, using the following assumption.

– H_1: a text block presenting a mark of an itemized list contains only one character of a specified set (bullet, square, etc.).

There are several ad-hoc heuristics we use to make a decision for each pair of text blocks to combine them or not. When a table is represented originally by a tagged form (e. g. as a table object in a Word-document) then PDF generators often store a logical reading order in printing text paragraph and cell content. We use the assumption that printing instructions forming a text inside each text or cell paragraph appear in the PDF file in the order that coincides with the human reading order of this text (Fig. 2, a–b). This feature can be especially useful to recover multi-line cell paragraphs without explicit borders. We also suppose that when ruling lines including invisible ones (cursor traces) are placed between two text blocks, then these text blocks belong to two separated paragraphs (Fig. 2, c).

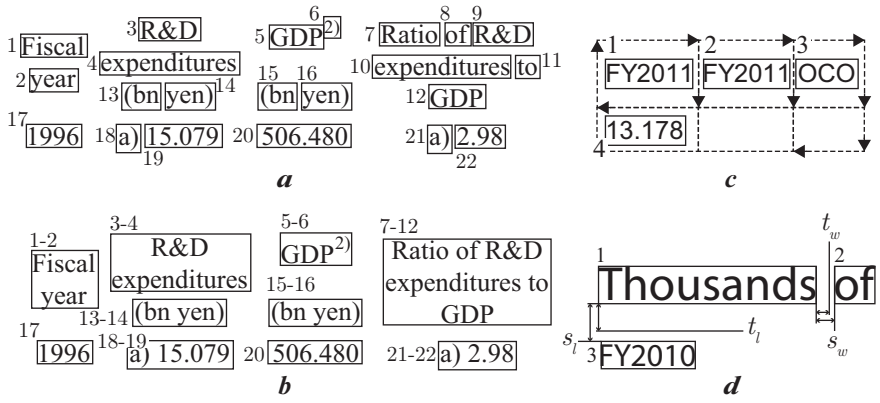

Fig. 2. Text block combination: words and their indices of the printing order (a) and cell paragraphs combined from the words (b); words are separated by the invisible ruling lines (cursor traces)(c); the word spacing (s_w) and line spacing (s_l) exceed the specified thresholds t_w and t_l respectively (d).

Two text blocks `tb1` and `tb2` can be combined into one `tb3` when they satisfy the specified assumption listed below:

- H_2, they are adjacent in the order of their appearance in the source PDF file: `tb1.order.end = tb2.order.start - 1`;
- H_3, there exist no ruling lines crossing `tb3.bbox`, the bounding box of the combined text block;
- H_4, they have identical fonts: `tb1.font = tb2.font` (including font family, size, bold, and italic attributes).

First, words are combined into text lines. The decision to combine a pair of text blocks `tb1` and `tb2` into one line `tb3` requires that they satisfy two additional assumptions:

- H_5, their word spacing, the horizontal distance between their bounding boxes (`tb2.xl - tb1.xr`), is less than a threshold depending on their width of space (`tb1.ws` or `tb2.ws`);
- H_6, their vertical projection intersection is more than a threshold.

Second, text lines are combined into paragraphs. Two text blocks `tb1` and `tb2` into one paragraph `tb3` when they satisfy two additional assumptions:

- H_7, their line spacing, the vertical distance between their bounding boxes (`tb2.yt - tb1.yb`), is less than a threshold depending on their font size (`tb1.fs` or `tb2.fs`);
- H_8, their horizontal projection intersection is more than a threshold.

The ideal case consists in that each combined text block is a textual content of one paragraph or cell, i. e. each non-empty cell contains only one text block.

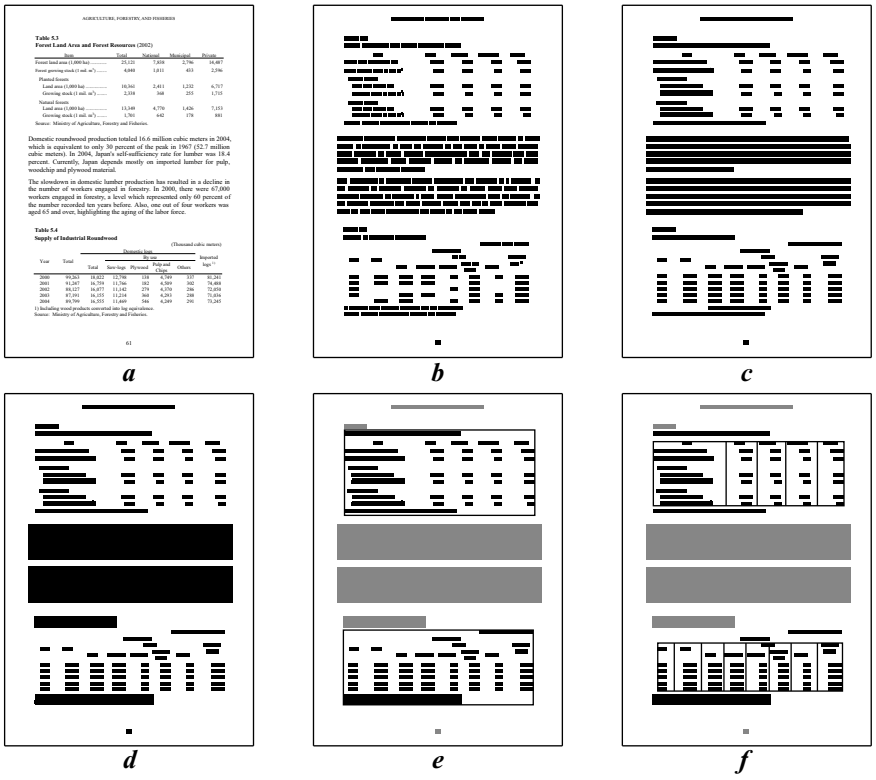

Fig. 3. The preliminary steps of the table detection: a source document (a); text blocks as words collected from "text positions" (b); text blocks as lines combined from words (c); text block as paragraphs combined from lines (d); table search areas located between table breaks (highlighted by gray color) (e); detected columns and tables (f).

2.3 Table Detection

TABBYPDF can detect both *bordered tables* (when borders presented explicitly as vector ruling lines) and *naked tables* (when their borders presented implicitly as invisible lines).

We assume that a bordered table is bounded by its outer borders represented by ruling lines. It also contains at least two rows and two columns completely bounded by ruling lines. To detect bordered tables, we select ruling lines that compose rectangular frames. When a frame contains or overlaps one or more others frames, then it is excluded from the processing. We search for groups of frames that intersect in their corner points. When a detected group of at least four frames composes at least two possible rows and two columns, then its outer ruling lines form a bounding box of a table.

Our approach to the naked table detection is based on separating a document into *table search areas*. A page can include one or more such areas (bounding

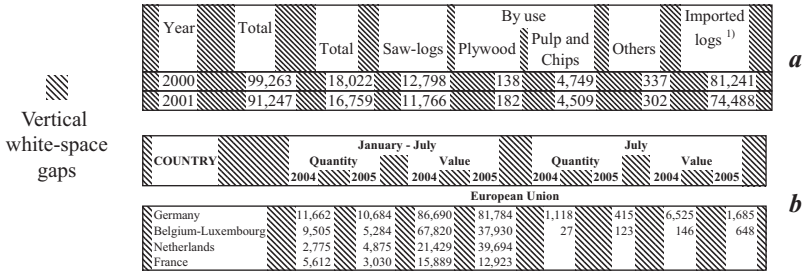

Fig. 4. The steps of the table detection in a table search area: combining neighbor rows agglutinated by their vertical white-space gaps into a region (a); combining neighbor regions agglutinated by their vertical white-space gaps into a table (b).

boxes) where tables can potentially be placed. We try to detect some *table breaks* (e. g. section headings, keywords like "Table", bitmap figures, multi-line and stretched paragraphs, page headers and footers, as well as detected bordered tables) on each page. We assume that a table search area cannot include any of the detected breaks. Thus, we detect each area located between two neighbor breaks (Fig. 3, e).

To detect naked tables we use a bottom-up segmentation of a search area: from simpler layout items to more complicated ones (Fig. 4). This bottom-up approach is inherited from our previous work on table detection in metafiles [20].

Text blocks are grouped into one row when they are in the transitive closure of the following relation: there exists an intersection of y-projections of two text blocks. Therefore, each extracted row contains two or more blocks. Moreover, there are no intersected rows. A white-space surrounding text blocks inside a row is segmented by our algorithm [22]. We select *vertical gaps* from the obtained white-space segments. They can be considered as implicitly cell borders. We group neighbor rows agglutinated by their white-space into one *table region* (Fig. 4, a) when they satisfy the following assumptions: (i) the vertical distance between them does not exceed a specified threshold based on an inter-line interval used in its source document; (ii) the vertical gaps of both rows correlate with each other in their x-projections (i. e. there is at least one vertical gap of the lower row that is correlated with a gap of the upper row).

Detected neighbor regions can be combined into one table when their vertical white-space gaps also correlate with each other in their x-projections (Fig. 4, b). Additionally, we construct table columns using well-aligned blocks. When a detected table intersects a column of well-aligned blocks that goes beyond its horizontal borders, then we extend its upper and bottom borders by the column. Moreover, when there is a signal word indicating a start of a table (e. g. "Table"), then we extend the upper border of the table to the bottom border of the text block bounding this signal word.

2.4 Table Structure Recognition

In this step, we construct rows and columns that constitute an arrangement of cells. The system provides two algorithms for slicing an inner space of a table into rows and columns. They both are presented more detail in our previous work [23].

The first (A_1) is based on the whitespace analysis. We use the algorithm [22] to recover horizontal and vertical gaps between text blocks. Each whitespace gap corresponds to a ruling. Thus, we try to recover all rulings, which separate cells in a table.

The second (A_2) is the analysis of connected text blocks. To generate columns, we first exclude each multi-column text block located in more than one column. We decide that a text block is multi-column when its horizontal projection intersects with the projections of two or more text blocks located in the same line. Each column is considered as an intersection of horizontal projections of one-column text blocks. Similarly, rows are constructed from vertical projections of one-row text blocks.

In this step, we also recover empty cells. Some of them can be erroneous, i. e. they absent in the source table. The system provides the ad-hoc heuristic to dispose of erroneous empty cells:

- H_9, *cell singleton*: if a column contains only one non-empty cell then the column is merged with the nearest column to the left.

3 Implementation

We develop a web-based prototype of TabbyPDF as a proof-of-concept for the presented approach. Its client-server architecture is shown in Fig. 5. The client part provides a web user interface built with Semantic UI[6]. It uses Mozilla PDF.js[7] tool for rendering PDF documents. The server part is a application based on Spring Framework[8]. It runs on Apache Tomcat[9] Java servlet container and stores data in the built-in database implemented by Apache Derby[10].

The client interacts with the server through REST-requests. Our RESTful API (Application Programming Interface) provides three requests:

- `/api/upload` POST-request uploads a PDF document specified in `file` parameter and returns positions of detected table bounding boxes in JSON format.
- `/api/extract` POST-request extracts tables from bounding boxes (positions in JSON format) specified in `data` parameter and returns extracted table cell structures in JSON format.

[6] `https://semantic-ui.com`
[7] `https://mozilla.github.io/pdf.js`
[8] `https://spring.io`
[9] `http://tomcat.apache.org`
[10] `https://db.apache.org/derby`

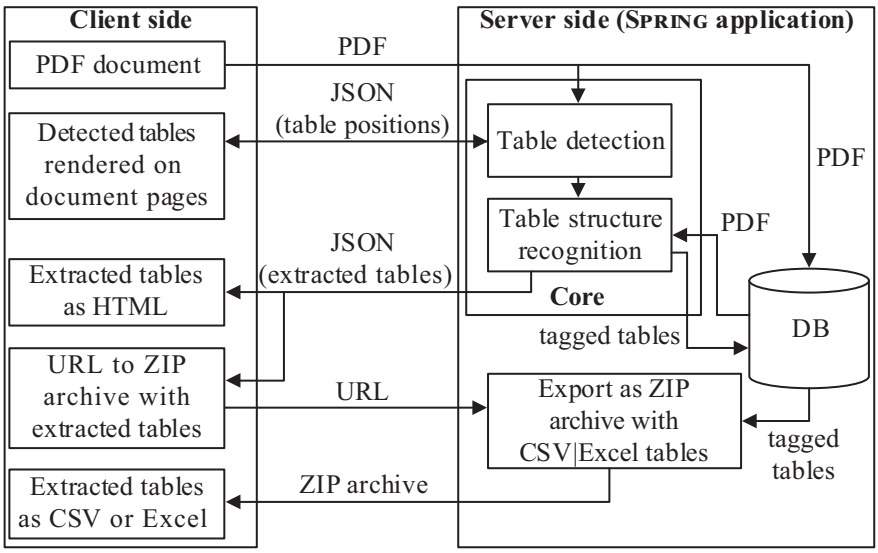

Fig. 5. The architecture of the web-based prototype of TabbyPDF.

– `/api/download/:id` GET-request builds ZIP-archive with extracted tables in HTML and CSV formats by an associated identifier provided in `id` parameter, and starts downloading this archive.

The workflow includes the following steps. A user selects a PDF document on the client-side. The client uploads it to the server. The core detects tables in the PDF document on the server-side. The server responds to the user, returning detected table positions to the client. The client displays the detected table areas over rendered PDF pages for the user. The user can correct the table positions on the client-side. The client sends the user correction data to the server. The core extracts tables from the specified areas on the server-side. The server returns extracted tables to the client for viewing results by the user. The user can download all extracted data stored in CSV and HTML formats.

Our work is an ongoing open-source project published on GitHub[11]. Its current state is viewable online[12].

4 Performance Evaluation

To evaluate the performance of our system we use the methodology for algorithms for table understanding in PDF documents proposed in the paper [5].

[11] TabbyPDF core: `https://github.com/cellsrg/tabbypdf`
TabbyPDF client: `https://github.com/cellsrg/tabbypdf-front`
TabbyPDF server: `https://github.com/cellsrg/tabbypdf-web`
[12] `http://cells.icc.ru/pdfte`

We also use the existing competition dataset[13], "ICDAR 2013 Table Competition" [4]. It contains 156 tables in 67 PDF documents collected from EU and US government websites. The evaluation was performed automatically using Nurminen's Python scripts[14] for comparing ground-truth and result files that implement this methodology with slight modifications.

The evaluated prototype of our system uses the iText[15] library for PDF interpretation to extract PDF objects from source documents and to generate initial text blocks and ruling lines. We use the threshold settings presented in the paper [23] as the C_2-configuration. The process of table extraction from all documents of this dataset performed by CPU (6M cache, up to 3.40 GHz) takes total 6342 ms: 3968 ms for the data preparation stage, 1563 ms for the table detection stage, and 810 ms for the cell structure recognition.

Table 1. Experimental results

Scores	Detection Phase	Recognition Phase	Table Extraction
Per-document averages			
Recall	0.8172	0.9233	0.8298
Precision	0.7605	0.9499	0.8339
F-score	0.7878	0.9364	0.8318

Table 2. Comparison with others in the table extraction

Tools	Per-document averages		
	Recall	*Precision*	*F-score*
FineReader	0.8835	0.8710	0.8772
OmniPage	0.8380	0.8460	0.8420
Nurminen [13]	0.8078	0.8693	0.8374
TabbyPDF	0.8298	0.8339	0.8318
Acrobat	0.7262	0.8159	0.7685
Nitro	0.6793	0.8459	0.7535
Silva [25]	0.7052	0.6874	0.6962
Yildiz at al. [27]	0.5951	0.5752	0.5850

The experimental results are shown in Table 1. They are expressed by the standard metrics in the information extraction: *recall*, *precision*, and *F-score*. The evaluation methodology is presented in the papers [4, 5] in detail. Briefly, these metrics are calculated (i) for both phases: the table detection and the structure recognition, separately, and (ii) for the fully automatic two-phase process of the table extraction. The methodology first calculates *recall* and *precision*

[13] `http://www.tamirhassan.com/dataset.html`
[14] `http://tamirhassan.com/competition/dataset-tools.html`
[15] `https://sourceforge.net/projects/itext`

scores for each document separately and then calculates average values based on the document scores.

The experimental results are comparable with the state-of-the-art academic and some commercial solutions (Table 2). The comparison with others (Table 2) are based on the data presented in the paper [4]).

5 Conclusion and Further Work

The main contribution of this work consists in demonstrating experimentally new possibilities of using the order of text printing commands as well as the cursor motion commands presented in PDF files to efficiently detect and extract untagged tables. We have formulated a set of valuable ad-hoc heuristics using these possibilities that provide combining words into text and cell paragraphs. This often allows to reduce errors of the table extraction.

The main advantage of our approach is that it can be quickly adapted to the table extraction from various domain-specific PDF documents (such as financial statements, business credit assessments, material safety data sheets, etc.) with a rich tabular content. Additional ad-hoc heuristics can be handcrafted and tuned for the target domain. This does not require to prepare training datasets that can be a costly process. The main disadvantage of this approach consists in that it is unsuitable when a PDF document is rasterized or contains glyphs without appropriate mappings to a standard encoding (e.g. Unicode).

The web-based prototype of TabbyPDF enables both automatic table detection and automatic cell structure recognition. It exports extracted tables in the editable format, CSV or HTML. Its output can be used as input in our rule-based spreadsheet data extraction system for transforming extracted arbitrary tables to relational ones [21, 24].

The further work is in progress on expanding the set of ad-hoc heuristics. The table extraction based on PDF content analysis requires an advance in document layout analysis (e.g. page header and footer detection; page column segmentation; title, section, and paragraph detection). We believe the involvement of the additional document layout analysis will allow improving our system. We also expect an advancement in involving ruling lines represented both PDF rendering instructions and ASCII pseudographics. This will provide extracting bordered tables more accurately.

6 Acknowledgments

This work is supported by the Russian Foundation for Basic Research (grants 18-07-00758 and 17-47-380007). The prototype of TabbyPDF is deployed on resources of the Shared Equipment Center of Integrated Information and Computing Network for Irkutsk Research and Educational Complex[16].

[16] `http://net.icc.ru`

References

1. Burdick, D., Evfimievski, A., Krishnamurthy, R., Lewis, N., Popa, L., Rickards, S., Williams, P.: Financial analytics from public data. In: Proc. Int. Workshop on Data Science for Macro-Modeling. pp. 4:1–4:6. DSMM'14 (2014), http://doi.acm.org/10.1145/2630729.2630742
2. Corrêa, A.S., Zander, P.O.: Unleashing tabular content to open data: A survey on pdf table extraction methods and tools. In: Proc. 18th Int. Conf. on Digital Government Research. pp. 54–63 (2017), http://doi.acm.org/10.1145/3085228.3085278
3. Coüasnon, B., Lemaitre, A.: Handbook of Document Image Processing and Recognition, chap. Recognition of Tables and Forms, pp. 647–677 (2014), http://dx.doi.org/10.1007/978-0-85729-859-1_20
4. Göbel, M., Hassan, T., Oro, E., Orsi, G.: ICDAR 2013 table competition. In: Proc. 12th Int. Conf. on Document Analysis and Recognition. pp. 1449–1453 (2013)
5. Göbel, M., Hassan, T., Oro, E., Orsi, G.: A methodology for evaluating algorithms for table understanding in PDF documents. In: Proc. 2012 ACM Symposium on Document Engineering. pp. 45–48 (2012), http://doi.acm.org/10.1145/2361354.2361365
6. Göbel, M., Hassan, T., Oro, E., Orsi, G., Rastan, R.: Table modelling, extraction and processing. In: Proc. 2016 ACM Symposium on Document Engineering. pp. 1–2 (2016), http://doi.acm.org/10.1145/2960811.2967173
7. Govindaraju, V., Zhang, C., R, C.: Understanding tables in context using standard NLP toolkits. In: Proc. 51st Annual Meeting of the Association for Computational Linguistics. p. 658664 (2013)
8. Hassan, T., Baumgartner, R.: Table recognition and understanding from PDF files. In: Proc. 9th Int. Conf. on Document Analysis and Recognition - Vol. 02. pp. 1143–1147 (2007), http://dl.acm.org/citation.cfm?id=1304596.1304833
9. Hu, J., Liu, Y.: Analysis of Documents Born Digital, pp. 775–804 (2014), http://dx.doi.org/10.1007/978-0-85729-859-1_26
10. Khusro, S., Latif, A., Ullah, I.: On methods and tools of table detection, extraction and annotation in PDF documents. J. Inf. Sci. 41(1), 41–57 (2015), http://dx.doi.org/10.1177/0165551514551903
11. Liu, Y., Bai, K., Mitra, P., Giles, C.L.: TableSeer: Automatic table metadata extraction and searching in digital libraries. In: Proc. 7th ACM/IEEE Joint Conf. on Digital Libraries. pp. 91–100 (2007), http://doi.acm.org/10.1145/1255175.1255193
12. Nganji, J.T.: The portable document format (PDF) accessibility practice of four journal publishers. Library & Information Science Research 37, 254–262 (2015), http://www.sciencedirect.com/science/article/pii/S0740818815000134
13. Nurminen, A.: Algorithmic extraction of data in tables in PDF documents. Master's thesis, Tampere University of Technology, Tampere, Finland (2013)
14. Oro, E., Ruffolo, M.: PDF-TREX: An approach for recognizing and extracting tables from PDF documents. In: Proc. 10th Int. Conf. on Document Analysis and Recognition. pp. 906–910 (2009)
15. Perez-Arriaga, M.O., Estrada, T., Abad-Mota, S.: TAO: system for table detection and extraction from PDF documents. In: Proc. 29th Int. Florida Artificial Intelligence Research Society Conference. pp. 591–596 (2016)
16. Ramel, J.Y., Crucianu, M., Vincent, N., Faure, C.: Detection, extraction and representation of tables. In: Proc. 7th Int. Conf. on Document Analysis and Recognition. pp. 374–378 vol.1 (2003)

17. Rastan, R., Paik, H.Y., Shepherd, J.: Texus: A task-based approach for table extraction and understanding. In: Proc. 2015 ACM Symposium on Document Engineering. pp. 25–34 (2015), http://doi.acm.org/10.1145/2682571.2797069
18. Rastan, R., Paik, H.Y., Shepherd, J.: A pdf wrapper for table processing. In: Proc. 2016 ACM Symposium on Document Engineering. pp. 115–118 (2016), http://doi.acm.org/10.1145/2960811.2967162
19. Sabol, V., Tschinkel, G., Veas, E., Hoefler, P., Mutlu, B., Granitzer, M.: Discovery and visual analysis of linked data for humans. In: The Semantic Web – ISWC 2014. pp. 309–324 (2014), https://doi.org/10.1007/978-3-319-11964-9_20
20. Shigarov, A., Bychkov, I., Ruzhnikov, G., Khmel'nov, A.: A method of table detection in metafiles. Pattern Recognition and Image Analysis 19(4), 693–697 (2009), https://doi.org/10.1134/S1054661809040191
21. Shigarov, A.: Table understanding using a rule engine. Expert Systems with Applications 42(2), 929–937 (2015)
22. Shigarov, A., Fedorov, R.: Simple algorithm page layout analysis. Pattern Recognition and Image Analysis 21(2), 324–327 (2011), http://dx.doi.org/10.1134/S1054661811021008
23. Shigarov, A., Mikhailov, A., Altaev, A.: Configurable table structure recognition in untagged pdf documents. In: Proc. 2016 ACM Symposium on Document Engineering. pp. 119–122 (2016), http://doi.acm.org/10.1145/2960811.2967152
24. Shigarov, A.O., Mikhailov, A.A.: Rule-based spreadsheet data transformation from arbitrary to relational tables. Information Systems 71, 123–136 (2017), https://doi.org/10.1016/j.is.2017.08.004
25. e Silva, A.C.: Parts that add up to a whole: a framework for the analysis of tables. Ph.D. thesis, University of Edinburgh, Tampere, Finland (2010)
26. e Silva, A.C., Jorge, A.M., Torgo, L.: Design of an end-to-end method to extract information from tables. International Journal of Document Analysis and Recognition (IJDAR) 8(2), 144–171 (2006)
27. Yildiz, B., Kaiser, K., Miksch, S.: pdf2table: A method to extract table information from PDF files. In: Proc. 2nd Indian Int. Conf. on Artificial Intelligence, Pune, India. pp. 1773–1785 (2005)