Towards High-Quality Text Stream Extraction from PDF

Technical Background to the ACL 2012 Contributed Task

Øyvind Raddum Berg, Stephan Oepen, and Jonathon Read
Department of Informatics, Universitetet i Oslo
{oyvinrb|oe|jread}@ifi.uio.no

Abstract

Extracting textual content and document structure from PDF presents a surprisingly (depressingly, to some, in fact) difficult challenge, owing to the purely display-oriented design of the PDF document standard. While a variety of lower-level PDF extraction toolkits exist, none fully support the recovery of original text (in reading order) and relevant structural elements, even for so-called born-digital PDFs, i.e. those prepared electronically using typesetting systems like LaTeX, OpenOffice, and the like. This short paper summarizes a new tool for high-quality extraction of text and structure from PDFs, combining state-of-the-art PDF parsing, font interpretation, layout analysis, and TEI-compliant output of text and logical document markup.[†]

1 Introduction—Motivation

To view a collection of scholarly articles like the ACL Anthology as a structured knowledge base substantially transcends a naïve notion of a *corpus* as a mere collection of running text. Research literature is the result of careful editing and typesetting and, thus, is organized around its complex internal structure. Relevant structural elements can comprise both *geometric* (e.g. pages, columns, blocks, or tables) and *logical* units (e.g. titles, abstracts, headings, paragraphs, or citations)—where (ideally) geometric and logical document structure play hand in hand to a degree that can make it hard to draw clear dividing lines in some cases (e.g. in itemized or numbered lists).

To date, the dominant standard for electronic document archival is *Portable Document Format* (PDF), originally created as a proprietary format by Adobe Systems Incorporated in the early 1990s and subsequently made an open ISO standard (which was officially adopted in 2008 and embraced by Adobe through a public license that grants royalty-free usage). PDF is something of a composite standard, unifying at least three basic technologies:

1. A subset of the PostScript page 'programming' language, dropping constructs like loops and branches, but including all graphical operations to draw layout elements, text, and images.

2. A font embedding system which allows a document to 'carry along' a broad variety of fonts (in various formats), as may be needed to ensure display just as the document was designed.

3. A structured storage system, which organizes various data objects—for example images and fonts—inside a PDF document.

All data objects in a PDF file are represented in a visually-oriented way, as a sequence of operators which—when interpreted by a PDF renderer—will draw the document on a page canvas. This is a natural approach considering the design roots of PDF as a PostScript successor and its original central role in desktop publishing applications; but the implications of such visually-centered design are unfortunate for the task of recovering textual content and logical document structure.

Interpretation of PDF operators will provide one with all the individual characters, as well as their formatting and position on the page. However, they generally do not convey information about higher level text units such as tokens, lines, or columns—information about boundaries between such units is only available implicitly through whitespace, i.e. the

[†]We are indebted to Rebecca Dridan, Ulrich Schäfer, and the ACL workshop reviewers for helpful feedback on this work.

Proceedings of the ACL-2012 Special Workshop on Rediscovering 50 Years of Discoveries, pages 98–103,
Jeju, Republic of Korea, 10 July 2012.

mere absence of textual or graphical objects. Furthermore, data fragments comprising content text on a page may consist of individual characters, parts of a word, whole lines, or any combination thereof—as dictated by font properties and kerning requirements. Complicating text extraction from PDF further, there are no rules governing the order in which content is encoded in the document. For example, to produce a page with a two-column layout, the page could be drawn by first drawing the first lines of the left and right columns, then the second lines, etc. Obtaining text in logical reading order, however, obviously requires that the text in the left column be processed before the one on the right, so a naïve approach to text extraction based on the sequencing of objects in the PDF file might produce undesirable results.

Since the standard is now open and free for anyone to use, we are fortunate to have several mature, open-source libraries to handle low-level parsing and manipulation of objects in PDF documents. For this project, we build on Apache PDFBox[1], for its maturity, relatively active support, and interface flexibility. Originally as an MSc project in Computer Science (Berg, 2011), we have developed a parameterizable toolkit for high-quality text and structure extraction from born-digital PDFs, which we dub PDFExtract.[2] In this application, we seek to approximate this structure by using all the visual clues and information we have available.

The data presented in a PDF file consists of streams of objects; by placing hardly any significance on the order of elements within these streams, and more on the visual result obtained by (virtually) 'rendering' PDF operations, the task of text and structure extraction is shifted slightly—from what traditionally amounts to stream-processing, and towards a point of view related to *computer vision*.

This view, in fact, essentially corresponds to the same problem tackled by OCR software, though without the need to perform actual character recognition. Some of the key elements of PDFExtract, thus, build on related OCR techniques and adapt and extend these to the PDF processing task. The process of 'understanding' a PDF document in this context is called document layout analysis, a task which is commonly treated as two sequential subprocesses. First, a page image is subjected to *geometric* layout analysis; the result of this first stage then serves as input for a subsequent step of *logical* layout analysis and content extraction. The following sections briefly review core aspects of the design and implementation of PDFExtract, ranging from low-level whitespace detection (§2), over geometric and logical layout analysis (§3 and §5, respectively), to aspects of font handling (§4).

2 Whitespace Detection

As a prerequisite to all subsequent analysis, segment boundaries between tokens, lines, columns, and other blocks of content need to be made explicit. Such boundaries are predominantly represented through whitespace, which is not overtly represented among the data objects in PDF files. The approach to whitespace detection and page segmentation in PDFExtract is an extension of the framework proposed by Breuel (2002) (originally in the context of OCR).

The first step here is to find a cover of the background whitespace of a document in terms of maximal empty rectangles. This is accomplished in a top-down procedure, using a whole page as its starting point, and working in a way abstractly analogous to quicksort or branch and bound algorithms. Whitespace rectangles are identified in order of decreasing 'quality' (as determined by size, shape, position, and relations to actual page content), which means that the result will in general be globally optimal—in the sense that no other (equal-sized) sequence of covering rectangles would yield a larger total quality sum.

Figure 1 illustrates the main idea of the algorithm, which starts from a bound (initially the page at large) and a set of non-empty rectangles, called *obstacles*. If the set is empty, it means that the bound is a maximal rectangle with respect to other obstacles (surrounding the bound). If, as in Figure 1, there are obstacles, the bound needs to be further subdivided. To this end, we choose one obstacle as a *pivot*, which ideally is centered somewhere around the middle of the bound. As no maximal rectangle can contain obstacles, in particular not the pivot, there are four possibilities for the solution of the maximal whitespace

[1] See `http://pdfbox.apache.org/` for details.

[2] See `http://github.com/elacin/PDFExtract/`.

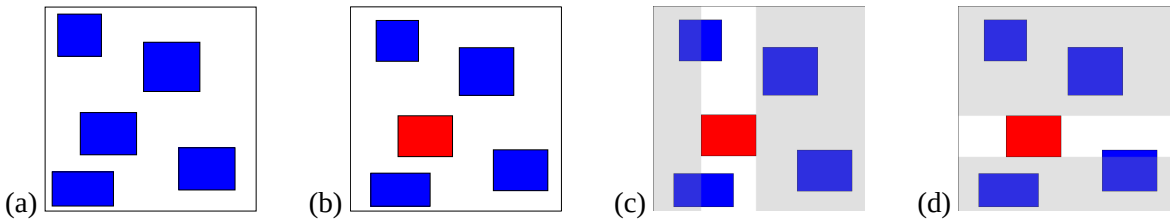

Figure 1: Schematic example of *one iteration* of the whitespace covering algorithm. In (a) we see some obstacles (in blue) contained within a bounding rectangle; in (b) one of them is chosen as as pivot (in red); and (c) and (d) show how the original bound is divided into four smaller rectangles (in grey) around the pivot.

rectangle problem—one for each side of the pivot. The areas of these four sub-bounds are computed, a list of intersecting obstacles is computed for each of them, and they are processed in turn.

As originally proposed by Breuel (2002), the basic procedure proved applicable to born-digital PDFs, though leaving room for improvements both in terms of the quality of results and run-time performance. Some deficiencies that were observed in processing documents from the ACL Anthology (and other samples of scholarly literature) are exemplified in Figure 2, relating to smallish, 'stray' whitespace rectangles in the middle of otherwise contiguous segments (top row in Figure 2), challenges related to relative differences in line spacing (middle), and spurious vertical boundaries introduced by so-called *rivers*, i.e. accidental alignment of horizontal spacing across lines (bottom). Besides adjustments to the rectangle 'quality' function, the problems were addressed by (a) allowing a small degree

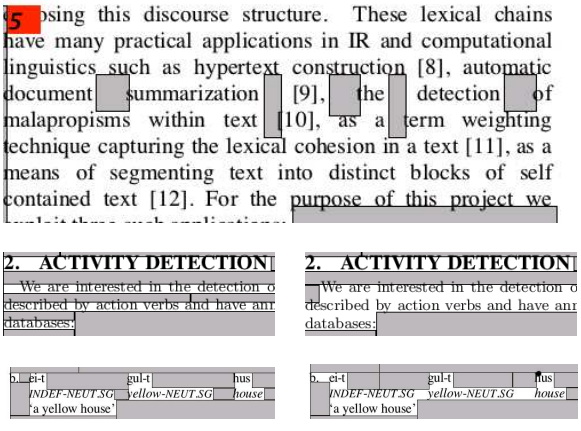

Figure 2: Select challenges to whitespace covering approach: stray whitespace inbetween groups of text (top); inter- vs. intra-paragraph spacing (middle); and 'rivers' leading to spurious vertical boundaries (bottom).

of overlap between whitespace rectangles and obstacles, (b) a strong preference for contiguous areas of whitespace (thus making the procedure work from the page borders inwards), (c) variable lower bounds on the height and width of whitespace rectangles, computed dynamically from font properties of surrounding text, and (d) a small number of specialized heuristic rules, to block unwanted whitespace rectangles in select configurations. Berg (2011) provides full details for these adaptations, as well as for algorithmic optimizations and parameterization that enable run-time throughputs of tens of pages per cpu second.

3 Determining Page Layout

The high-level goal in analyzing page layout is to produce a hierarchical representation of a page in terms of *blocks* of homogenous content, thus making explicit relevant spatial relationship between them. In the realm of OCR, this task is often referred to as *geometric layout analysis* (see, for example, (Cattoni et al., 1998)), whereas the term *(de)boxing* has at times been used in the context of text stream extraction from PDFs. In the following paragraphs, we will focus on column boundary detection, but PDFExtract essentially applies the same general techniques to the identification of other relevant inter-segment boundaries.

While whitespace rectangles are essential to column boundary identification, there is of course no guarantee for the existence of *one* rectangle which were equivalent to a whole column boundary. First, as a natural consequence of the whitespace detection procedure, horizontal rectangles can 'interrupt' candidate colum boundaries. Second, there may well be typographic imperfections causing gaps in the identified whitespace (as exemplified in the top of Fig-

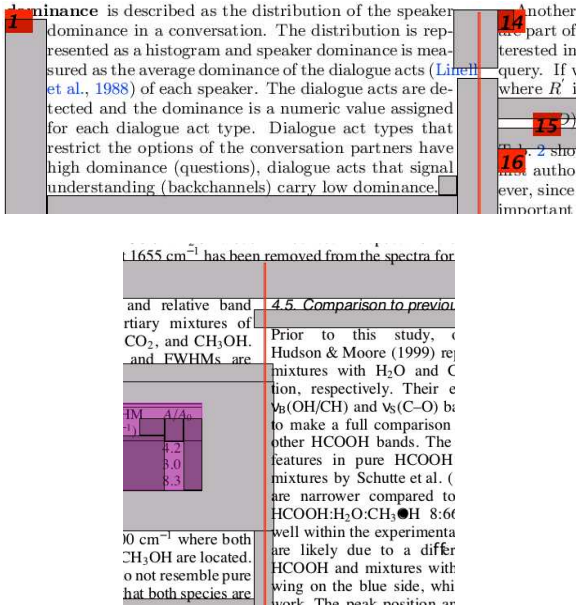

Figure 3: Select challenges to column identification: text elements protruding into the margin (top) and gaps in whitespace rectangle coverage (often owed to processing bounds imposed for premium performance).

ure 3), or it can be the case that geometric constraints or computational limits imposed on the whitespace cover algorithm result in 'missing' whitespace rectangles (in the bottom of Figure 3). Whereas the original design of Breuel (2002) makes no provisions for these cases, PDFExtract adapts a revised, three-step approach to column detection, viz. (a) extracting an initial set of candidate boundaries; (b) heuristically expanding column boundary candidates vertically; and (c) combining logically equivalent boundaries and filtering unwarranted ones. Here, both steps (a) and (b) assume geometric constraints on the aspect ratio of candidate column boundaries, as well as on the existence and relative proportions of surrounding non-whitespace content. Again, please see Berg (2011) for further background on these steps.

With column boundaries in place, PDFExtract proceeds to the identification of *blocks* of content (which may correspond to, for example, logical paragraphs, headings, displayed equations, tables, or graphical elements). This step, essentially, is realized through a recursive 'flooding' function, forming connected blocks from adjacent, non-whitespace PDF data objects where there are no intervening whitespace rectangles. Regions that (by content or font properties) can be identified as (parts of) mathematical equations receive special attention at this stage, allowing limited amounts of horizontally separating whitespace to be ignored for block formation. In a similar spirit, line segmentation (i.e. grouping of vertically aligned data objects) is performed block-wise—sorting content within each block by Y-coordinates and determining baselines and inter-line spacing in a single downwards pass.

The final key component in geometric layout analysis is the recovery of reading order (recalling that PDFs do not provide reliable sequencing information for data objects). PDFExtract adapts one of the two techniques suggested by Breuel (2003), viz. topological sorting of lines (which can include single-line blocks, where no block-internal line segmentation was detected) based on (a) relations of hierarchical nesting and (b) relative geometric positions. PDFExtract was tested against a set of some 100 diverse PDF documents (from different sources of scholarly literature, a range of distinct PDF generators, quite variable layout, and multiple languages), and its topological content sorting (detailed further in Berg, 2011) was found to give very satisfactory results in terms of reading order recovery.

4 Font Handling and Word Segmentation

Many of the steps of geometric layout analysis outlined above depend on accurate coordinate information for glyphs, which turned out an unforeseen low-level challenge in our approach of building PDFExtract on top of Apache PDFBox. Figure 4 (on the left) shows a problematic example of 'raw' glyph placement information. Several factors contribute to incorrect glyph positioning, including the sheer variety of font types supported in PDFs, missing information about non-standard, embedded fonts, and design limitations and bugs in PDFBox. To work around common issues, PDFExtract includes a couple of patches to PDFBox internals as well as specialized code for different types of font embedding in PDF to perform boundary box computation, position offsetting, and and mapping to Unicode code points. The (much improved though not quite perfect) result of these adjustments, when applied to our running example, is depicted in the middle of Figure 4.

With the ultimate goal of creating a high-quality

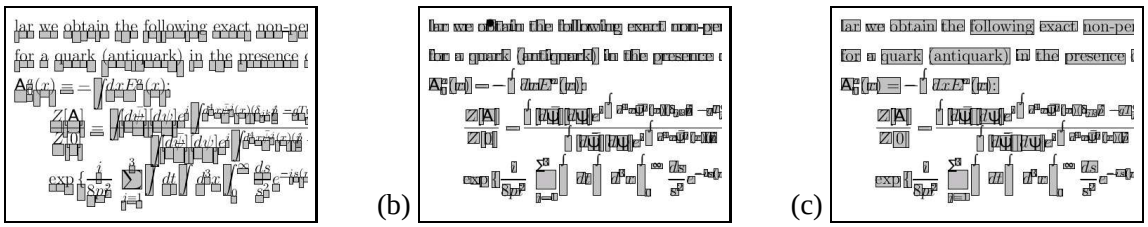

Figure 4: Examples of font-related challenges (before and after correction) and word segmentation.

(structured) text corpus from ACL Anthology documents, *word segmentation* naturally is a mission-critical component of PDFExtract. Seeing that inter-word whitespace is more often than not *omitted* from PDF data objects, word segmentation—much like other sub-tasks in geometric layout analysis—operates in terms of display positions. Determining whether the distance between two adjacent glyphs represents a word-separating whitespace or not, might sound simple—but in practice it proved difficult to devise a generic solution that performs well across differences in fonts and sizes (and corresponding variation in kerning, i.e. intra-word spacing), deals with both high-quality and poor typography, and is somewhat robust to remaining inaccuracies in glyph positions . PDFExtract arrived at a novel algorithm that approximates character text spacing (as could be set by the PDF T_c operator) by averaging a selection of the smaller character distances within a line. The resulting average character spacing is subsequently used to *normalize* horizontal distances, i.e. subtract line-specific character spacing from every distance on that line—to ideally center character distances around zero, while leaving word distances larger (they will also be relatively much larger than before in comparison). The identification of word boundaries itself, accordingly, becomes straightforward, comparing normalized distances to a percentage of the local font size. The results of this process are shown for our example in the right of Figure 4.

5 (Preliminary) Logical Layout Analysis

In our view, thorough geometric layout analysis is an important prequisite of logical layout analysis. Hence, the emphasis of Berg (2011) was with respect to the geometric analysis. However, what follows is an overview of the preliminary procedure in PDFExtract to determine logical document structure from geometric layout and typographic information.

The process begins by collating a set of text *styles* (i.e. unique combinations of font type and size). Then, various heuristics govern the assignment of styles to logical roles:

Body text Choose whichever style occurs most frequently (in terms of the number of characters).

Title Choose the header-like block on the first page that has the largest font size.

Abstract If one of the first pages has a single-line block with a style which is bigger or bolder than body text, and contains the word abstract, it is chosen as an abstract header. All body text until the next heading is the abstract text.

Footnote Search for blocks on the lower part of the page that are smaller than body text; check that they start with a number or other footnote-indicating symbol.

Sections Identify section header styles by compiling a list of styles that are either larger than or have some emphasis on the body text style, and have instances with evidence of section numbering (e.g. *1.1*, *(1a)*). Infer the nesting level of each section header style from its order of occurrence in the document; a section heading will always appear earlier than a subsection heading, for instance.

Having identified the different components in the document, these are used to create a logical hierarchical representation following the TEI P5 Guidelines (TEI Consortium, 2012) as introduced by Schäfer et al. (2012). Title, abstract, floaters, and figures are separated from the main text. The body of the document is then collated into a tree of section elements, with headers and body text. Body text is collected by combining consecutive text blocks that

have identical styles, before inferring paragraphs on the basis of indented initial lines. Dehyphenation is tackled using a combination of a lexicon and a set of orthographic rules.

6 Discussion—Outlook

PDFExtract provides a fresh and open-source take on the problem of high-quality content and structure extraction from born-digital PDFs. Unlike existing initiatives (e.g. the basic `TextExtraction` class of PDFBox or the pdftotext command line utility from the Poppler library[3]), PDFExtract discards sequencing information available in the so-called PDF text stream, but instead applies and adapts techniques from OCR—notably a whitespace covering algorithm, column, block, and line detection, recovery of reading order based on line-oriented topological sort, and improved word segmentation taking advantage of specialized PDF font interpretation. While very comprehensive in terms of its geometric layout analysis, PDFExtract to date only make available a limited range of logical layout analysis functionality (and output into TEI-compliant markup), albeit also in this respect more so than pre-existing PDF text stream extraction approaches.

For the ACL 2012 Contributed Task on *Rediscovering 50 Years of Discoveries* (Schäfer et al., 2012), PDFExtract outputs for the born-digital subset of the ACL Anthology are a component of the 'starter package' offered to participants, in the hope that content and structure derived from OCR techniques (Schäfer & Weitz, 2012) and those extracted directly from embedded content in the PDFs will complement each other. As discussed in more detail by Schäfer et al. (2012), the two approaches have in part non-overlapping strengths and weaknesses, such that aligning content elements that correspond to each other across the two universes could yield a multi-dimensional, ideally both more complete and more accurate perspective. PDFExtract is a recent development and remains subject to refinement and extension. Beyond a limited quantitative and qualitative evaluation review by Berg (2011), the exact quality levels of text and document structure that it makes available (as well as relevant factors of variation, across different types of documents in the ACL Anthology) remains to be determined empirically.

We make available the full package, accompanied by some technical documentation (Berg, 2011), as well as a sample of gold-standard TEI-compliant target outputs) in the hope that it may serve as the basis for future work towards the ACL Anthology Corpus—both at our own sites (i.e. the University of Oslo and DFKI Saarbrücken) and collaborating partners. We would enthusiastically welcome additional collaborators in this enterprise and will seek to provide any reasonable assistance required for the deployment and extension of PDFExtract.

[3] See `http://poppler.freedesktop.org/`.

References

Berg, Ø. R. (2011). *High precision text extraction from PDF documents.* MSc Thesis, University of Oslo, Department of Informatics, Oslo, Norway.

Breuel, T. (2002). Two geometric algorithms for layout analysis. In *Proceedings of the 5th workshop on Document Analysis Systems* (pp. 687–692). Princeton, USA.

Breuel, T. (2003). Layout analysis based on text line segment hypotheses. In *Third international workshop on Document Layout Interpretation and its Applications.* Edinburgh, Scotland.

Cattoni, R., Coianiz, T., & Messelodi, S. (1998). *Geometric layout analysis techniques for document image understanding. A review* (ITC-irst Technical Report TR#9703-09). Trento, Italy.

Schäfer, U., Read, J., & Oepen, S. (2012). Towards an ACL Anthology corpus with logical document structure. An overview of the ACL 2012 contributed task. In *Proceedings of the ACL-2012 main conference workshop on Rediscovering 50 Years of Discoveries.* Jeju, Republic of Korea.

Schäfer, U., & Weitz, B. (2012). Combining OCR outputs for logical document structure markup. Technical background to the ACL 2012 Contributed Task. In *Proceedings of the ACL-2012 main conference workshop on Rediscovering 50 Years of Discoveries.* Jeju, Republic of Korea.

TEI Consortium. (2012, February). *TEI P5: Guidelines for electronic text encoding and interchange.* (http://www.tei-c.org/Guidelines/P5)