Josef B. Baker; Alan P. Sexton; Volker Sorge

Extracting Precise Data on the Mathematical Content of PDF Documents

In: Petr Sojka (ed.): Towards Digital Mathematics Library. Birmingham, United Kingdom, July 27th, 2008. Masaryk University, Brno, 2008. pp. 75--79.

Persistent URL: http://dml.cz/dmlcz/702535

Extracting Precise Data on the Mathematical Content of PDF Documents

Josef B. Baker, Alan P. Sexton, and Volker Sorge

School of Computer Science, University of Birmingham
E-mail: J.Baker@cs.bham.ac.uk, A.P.Sexton@cs.bham.ac.uk,
V.Sorge@cs.bham.ac.uk
URL: http://www.cs.bham.ac.uk/~jbb, http://www.cs.bham.ac.uk/~aps
http://www.cs.bham.ac.uk/~vxs

Abstract. As more and more scientific documents become available in PDF format, their automatic analysis becomes increasingly important. We present a procedure that extracts mathematical symbols from PDF documents by examining both the original PDF file and a rasterized version. This provides more precise information than is available either directly from the PDF file or by traditional character recognition techniques. The data can then be used to improve mathematical parsing methods that transform the mathematics into richer formats such as MathML.

Key words: document analysis, semantic analysis, PDF, MathML, mathematics

1 Introduction

In recent years, the PDF format has become widely accepted as a quasi-standard for document presentation and exchange. Document analysis starting directly from PDF documents is therefore becoming increasingly important. Research in this area currently ranges over extracting and identifying various components of the documents. The output can then be used for purposes such as improving search and indexing capabilities, or converting into other formats such as XML. Some of this work exploits information contained in the PDF file, whilst others only work with a rasterized version of the file. The former approach suffers from poor glyph boundary identification, necessary in mathematical formula recognition, while the latter loses potentially important data contained within the PDF such as precise discrimination between visually very similar characters (see [3,4]).

The goal of our work is the extraction of mathematical expressions from PDF documents, by accessing the PDF file directly. We achieve this by extracting information on the fonts and geometric positions of single characters from the PDF source (described in Section 2). Since the positional information in the PDF file is geared towards display of the document only, it is too imprecise for mathematical formula recognition. We augment our data with information

Petr Sojka (editor): DML 2008, Towards Digital Mathematics Library, pp. 75–79.
 ISBN 978-80-210-4658-0

gained from image analysis of a rasterized version of the same document (see Section 3). This results in precise information which is used to improve existing parsing techniques. While our work is currently restricted to PDF files generated from LaTeX source, containing type 1 fonts only, the technique is general enough to be transferable to PDF generated from other sources.

2 Extracting Content from PDF

To extract mathematical content from PDF files, we are mainly interested in identifying mathematical characters together with their geometric layout. Working with PDF files using Type 1 fonts, we extract a mapping of each symbol to a Unicode character, the origin and baseline of each character and a font bounding box. A font bounding box is the smallest box that contains every symbol within the font. The basic content extraction procedure consists of three steps: (1) content decompression, (2) font extraction, and (3) character extraction.

Content Decompression. PDF files are usually created in a compressed form, with much, but not all, of the content encoded using one or more compression algorithms. We use the open source Java application, Multivalent [2], to decompress the PDF files before further content analysis. The resulting file, while still in a valid PDF format that produces exactly the same output as the original, contains commands and content in an easily processable form. In particular, it contains hierarchically arranged information on layout, structure, appearance and other aspects of the document. For extraction of mathematical content, much of this is superfluous, and we concentrate mainly on information on fonts and characters.

Font Extraction. In order to extract the character information from the PDF file, two passes of the file are required. The first is used to collect font information, which is output as a list containing each font name, along with all of its corresponding characters and their font bounding box sizes. This information is collated by exploiting the following objects in the PDF document:

Resource Dictionaries: give the location of resources such as fonts and graphics states used by a page. One exists for each page in the PDF file.

Font Dictionaries: contain the addresses of the associated objects required for each font, such as encodings, widths and descriptors. They also contain details about the sets of characters to be used, allowing the mapping of byte values to their correct encoding and width.

Font Descriptors: contain various information about each font, including the character set used and the font bounding box.

Font Encodings: are essentially mappings from byte values to the specific character information in a font. In the case of Type 1 fonts, this character information is the name of the character. Specific font encodings are created for each document to capture the precise characters that are used.

Font Widths: are not the actual widths of characters. Instead they are the distances between the origin of a character and that of the following character, if written continuously on the same line. Not all of the characters in the font will have an entry; only those actually used within the file. Therefore they have to be carefully mapped to the correct characters, by using information from the font dictionary.

The information is extracted by finding each font from the resource dictionary, then gathering the required data while following the links to all of the appropriate objects.

Character Extraction. Once the font extraction is completed, we extract information on the actual characters and their positions in a second pass. Here we consider only the so called *Content Streams*, which, most importantly, contain the instructions for displaying and transforming text and in-line images.

Each time we enter a text environment within a content stream, the font and font size are both stated, followed by a command that transforms the current position and the byte values for each character to be displayed. When the procedure encounters the byte values, the appropriate character along with its font bounding box is obtained from our encoding list that was created in the previous stage. The font bounding box is then transformed, by scaling from PDF text space into its actual size in the document. This is completed by multiplying the bounding box values by the font size, then dividing by 1,000. The coordinates of the top left corner of the character, along with extensions of the x and y axis are then computed and output. The next set of byte values may or may not have a font name and size associated with them, but will always have position commands. Since these are relative to the previous position, we have to remember the state after the execution of the previous command, and also complete each command sequentially in order to obtain correct output.

In addition to regular characters, PDF lines can also contain in-line images which are straight lines that are often used to represent elements such as the tail on the radical symbol or a fraction's quotient line. We extract these lines in a similar manner to text, following the transformation to obtain the lines coordinates, but then using the line's length and weight to determine the exact vertical and horizontal extensions.

3 Calculation of Bounding Boxes

Although we now have exact information on the individual characters of the PDF file, we only have a rough idea about their position, as we only have information about the position of each character's font bounding box. However, font bounding boxes are generally much larger than the actual character's bounding box and we have no information about where exactly a character sits inside the font box. Moreover, font boxes often overlap horizontally and vertically or are even overlayed. While the font box data is sufficient to recognise simple text given in a line, more accurate data is required when

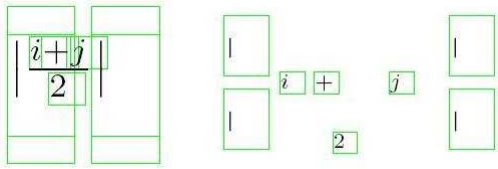

Fig. 1. Font bounding boxes before and after shrinking

trying to calculate 2D relationships between symbols, as necessary for formula recognition. To obtain the actual exact bounding boxes of characters and their positions, we use the following strategy:

1. Render the PDF document into an image.
2. Find all glyphs in the image and identify their minimal bounding boxes.
3. Register the discovered glyph bounding boxes from the image with the corresponding font boxes from the PDF document.
4. Compose bounding boxes for multi-glyph characters.

The PDF document is transformed into a tiff image, which is then systematically scanned for all connected areas of black pixels in order to identify the separate glyphs. Once all glyphs are found the bounding boxes are computed and output.

The next step is more complex. As the font bounding boxes can contain more than one character, it is difficult to match the minimal bounding boxes to the appropriate character. To overcome this, during the first pass of the PDF file, the font size of each character is decreased. This shrinks each character around its origin, still keeping its relative position. The shrinking prevents the overlapping of the font bounding boxes, which makes matching an easier task. We are currently employing a static shrinking factor of 0.1, a figure we have arrived at after some experimental testing. However, future work will includethe ability to dynamically determine an appropriate factor.

The first image in Figure 1 shows the font bounding boxes of the original file. It is noticeable that some of the boxes contain multiple characters, especially those of the delimiters, which are actually constructed from two separate symbols. The image on the right is after the shrinking has taken place, which, for ease of viewing, is not displayed to scale. The horizontal line does not need further analysis as the exact coordinates and bounding box is extracted directly from the PDF.

The i and j consist of two glyphs (for the base stroke and the dot above it) and thus will have two bounding boxes from the tiff analysis. However, as they are both contained within a single font bounding box they are matched to a single character with an appropriate bounding box containing both glyphs. In the case of the vertical delimiters, a single glyph is actually made up of multiple

PDF characters. This can occur with a number of characters in PDF, including other large delimiters and square roots. These are identified and reconstructed into a single character and bounding box. Matching all other characters is a trivial task.

Once all of the characters minimal bounding boxes have been identified, they are scaled back to their original size.

4 Conclusion

The advantages of combining information from both the original PDF file and a rasterized version is threefold. First, there is no error in identifying characters which can occur when using traditional character recognition methods. Second, we have precise bounding box information gained from the image analysis which is not available from the PDF. Finally, we have extra information about the characters, including font names, sizes, origins and baselines.

In addition, having the baseline information allows the grouping of sets of characters as a preprocessing step before further analysis takes place. For example consecutive letters making a string, in the case of sin, cos and ln, and consecutive digits making a larger integer. All this together gives us more precise information on the actual mathematical content, which we use as a basis for higher level formula recognition approaches such as projection profile cutting and graph grammars (see [6,5]). Our particular aim is thereby to turn primarily heuristic approaches into more analytic techniques.

While our current implementation is limited to PDF versions 1.4 or later, using Type 1 fonts, there exist methods for the conversion of fonts, and converting between different PDF versions, which we hope to exploit in the future (see [1]).

References

1. Proberts, S., Brailsford, D. Substituting Outline Fonts for Bitmap Fonts in Archived PDF Files. In *Soft. Pract Exper.*, 33(9) pp. 885–899, 2003.
2. Phelps, T. Multivalent. `http://multivalent.sourceforge.net/`.
3. Rahman, F., Alam, H. Conversion of PDF documents into HTML: A case study of document image analysis. In *Conf. on Signal, Systems, Computers* pp. 87–91, 2003.
4. Shao, M., Futrelle, R. Graphics recognition in PDF documents. In *Proc. of GREC 2005*, *LNCS* 3926. Springer, 2006.
5. Raja, A., Rayner, M., Sexton, A., Sorge, V. Towards a parser for mathematical formula recognition. In *Proc. of MKM 2006, LNCS* 4108. pp. 139–151, Springer, 2006.
6. Grbavec, A., Blostein, D. Mathematics recognition using graph rewriting. In *Proc. of ICDAR '95*, pp. 417–421, 1995.