Enhancing the Searchability of Page-Image PDF Documents Using an Aligned Hidden Layer from a Truth Text

Ian A. Knight
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
ian.knight.1990@gmail.com

David F. Brailsford
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
dfb@cs.nott.ac.uk

ABSTRACT

The search accuracy achieved in a PDF image-plus-hidden-text (PDF-IT) document depends upon the accuracy of the optical character recognition (OCR) process that produced the searchable hidden text layer. In many cases recognising words in a blurred area of a PDF page image may exceed the capabilities of an OCR engine.

This paper describes a project to replace an inadequate hidden textual layer of a PDF-IT file with a more accurate hidden layer produced from a 'truth text'. The alignment of the truth text with the image is guided by using OCR-provided page-image co-ordinates, for those glyphs that are correctly recognised, as a set of fixed location points between which other truth-text words can be inserted and aligned with blurred glyphs in the image. Results are presented to show the much enhanced searchability of this new file when compared to that of the original file, which had an OCR-produced hidden layer with no truth-text enhancement.

CCS Concepts

•**Applied computing** → *Document analysis; Document searching; Optical character recognition;*

Keywords

PDF, OCR, Tesseract, searchability, truth text

1. INTRODUCTION

In a previous paper [3] we presented the first stages of an effort to provide a computer-assisted framework for tagging key phrases within a variable-quality PDF document collection, followed by automated extraction and assembly of the tagged phrases into standardised summary documents. The corpus of 20,000 documents used for this work was that held by the Cochrane Schizophrenia Group's register of trials. This corpus is overwhelmingly archived as PDF documents.

DocEng '16, September 12 - 16, 2016, Vienna, Austria
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4438-8/16/09...$15.00
DOI: http://dx.doi.org/10.1145/2960811.2967157

As a first step in automating the extraction of qualitative and quantitative data it is vital that all PDF documents in the collection be as searchable as possible, so that key words and phrases can be highlighted by the PDF viewer.

PDF supports the rendering of arbitrarily complex text, in any chosen font, with diagrams being drawn using the correct arc, line and spline primitives. Bitmapped material such as photographs (either lossily or losslessly compressed) are handled by the PostScript/PDF **image** operator. PDF files of this quality are nowadays referred to as 'PDF—Formatted Text and Graphics' (PDF-FTG). The key advantage of this format is that text strings can usually be accurately located anywhere within the PDF file.

For the most part, the Cochrane collection contains papers of PDF-FTG quality. However, in the previous paper [3] we reported that fully one-third of the corpus was not in searchable format because PDF allows pages to be rendered simply as (unsearchable) bitmapped page-images (PDF-I).

Since 1994 an extra hybrid format has existed for PDF files, called 'PDF Image plus Hidden Text' (PDF-IT). here an invisible, searchable, text overlay is created for each page image. With careful choice of font, type size and inter-word spacing, in this hidden layer, it is possible to make it be in exact registration with the perceived words in the page image.

PDF-IT has the great virtue that searched-for words are highlighted by illuminating the correct bounding box in the hidden layer but this manifests itself as a highlight in the exactly superposed image layer, thereby creating the illusion of a textually searchable image.

Thus, by using OCR techniques to add a hidden, but searchable, text overlay to the 6,000 PDF-I documents in the Cochrane collection, an enhanced degree of searchability was achieved. However, in many cases poor quality source material (often from scanned-in papers) resulted in material which caused the OCR software enormous problems. It was decided to investigate whether the OCR process might be helped if assisted by a truth text for the papers in question.

1.1 Truth texts

A *truth text* (or *ground truth text*) is essentially a canonically correct version of the textual content of a paper. The characteristic of such a truth text is that it will contain the correct words in the correct order, but it will not reflect any of the typesetting or layout decisions made by the publisher in the final printed version. For example, paragraph breaks may be altered, hyphens may be introduced and text may be set in multiple columns or wrapped around figures. As a consequence, the process of *aligning* such a truth text with the PDF document produced from it is complicated.

1.2 Related work

While there is little work on the actual creation of PDF-IT documents in the manner we describe, there is a large body of work focused on aligning OCR results with a ground truth text for the purposes of training. We now give a description of some recent approaches to the alignment problem.

Several authors [5, 2, 6] describe methods for aligning ground truth with OCR results. These involve computing a transformation function that will map the image used as ground truth onto the OCR results as closely as possible, so that the results may be matched character for character. They presuppose that the ground truth exists in electronic form, generated from the same typesetting code used to produce the document undergoing OCR. We find their approaches to be very useful in terms of attempting to closely couple the words in the OCR results with their corresponding words in the truth text, but in our case the original code used for the typesetting will not be available to us.

Feng [1] and Yalniz [8] both present different methods of aligning ground truth with OCR that do not rely on the existence of a canonically correct electronic document. Instead, both assume their ground truth texts to be in the form of plain text, typically ASCII. Feng constructs a hidden Markov model of the words in the ground truth, using it to predict the ground truth for each of the OCR results, while Yalniz solves the problem recursively by dividing the texts into sections bounded by unique words, and repeatedly subdividing until the sections are small enough to be easily aligned word-for-word. These approaches both have advantages in that they do not presuppose any correspondence between the layout of the ground truth and the layout of the actual document: they assume the ground truth to contain only the correct words in the correct order. However, both approaches require a considerable amount of preprocessing of one or both texts in order to run. We wished to explore the possibility of using a similarly unformatted ground truth text, but without requiring the overhead of constructing a hidden Markov model or similar.

It is worth emphasising again that we are *not* concerned with training the OCR to better recognise words it has difficulty with; instead, we take the possibly flawed OCR output and use the truth text to establish the locations of corresponding word before inserting the corrected words into the document as a hidden layer.

1.3 Structure

The rest of the paper is organised as follows. Sec. 2 describes our method of constructing PDF-IT documents using OCR results from a PDF-I document and a truth text. Sec. 3 describes the performance of our method in terms of the accuracy of the corrected hidden text layer, and Sec. 4 concludes and suggests some avenues for future work.

2. PDF-IT CREATION

In this section we describe our method for constructing PDF-IT documents from pre-existing PDF-I/IT documents. The method is divided into three phases: generation of a first estimate of the hidden text layer using OCR, correction of the hidden text using the truth text, and inserting the corrected hidden text layer into the PDF. Each of these phases is described below.

2.1 OCR generation of hidden text

In the first phase we run the page images from the original PDF through an OCR engine in order to produce an initial estimate of the pages' contents. This was done even if the provided document was already PDF-IT, with some version of a hidden text layer already present.

The OCR engine we used was Tesseract, an open-source engine owned by Google that provides a very comprehensive C++ API. We made use of Tesseract rather than Adobe's built-in OCR engine for two reasons: firstly, Tesseract is freely available whereas access to Adobe's PDF APIs is expensive; and secondly, Tesseract makes it very easy to access not just the strings it identifies but also the bounding boxes of these strings. This makes the creation of a new hidden text layer at the end of the process much more straightforward. Tesseract's accuracy is broadly comparable to that of Acrobat Capture, so we felt confident in using Tesseract for our preliminary investigations.

2.2 Correction of hidden text

In the second phase, we correct the OCR output obtained previously using the provided truth text. The OCR results are here represented as a vector O of words $O_1, O_2, \ldots, O_n$, while the words obtained from the truth text are similarly represented as a vector T of words $T_1, T_2, \ldots, T_m$. We also constructed a BK-tree (see Sec. 2.2.3) of the words in the truth text, denoted $\mathrm{BK}(T)$, using edit distance as the distance metric.

Our approach consists of stepping through both vectors word by word, attempting to determine which word from the truth text best matches the current word in the OCR results. Word matching consisted of two phases: an attempt to find an *exact* match to the OCR word in the words near to our current position in the truth text; and an attempt to find the best *approximate* match in the entire truth text, if no exact match could be found. The method is summarised in Alg. 1. If an exact match was found, no correction is needed and iteration continues. If no exact match was found, then the best-guesses of ExactMatch and ApproximateMatch were compared, and the string closest to the OCR result was used to correct the OCR.

Algorithm 1 Method for OCR correction

```
1: $i \leftarrow 1; j \leftarrow 1$
2: while $i \leq n$ do
3:     $(j, m) \leftarrow \textsc{ExactMatch}(i, j)$
4:     if $m \neq 1$ then
5:         $j' \leftarrow \textsc{ApproximateMatch}(i, j)$
6:         if $d_A(O_i, T_{j'}, T_j) = 1$ then
7:             $j \leftarrow j'$
8:         end if
9:     end if
10:    $O_i \leftarrow T_j$
11:    $i \leftarrow i + 1; j \leftarrow j + 1$
12: end while
```

2.2.1 String distance metrics

When finding a match, two distance metrics were used: *Levenshtein (edit) distance*, d_L, and *Jaccard distance*, d_J.

The edit distance between two strings is the minimum number of character insertions, deletions, and substitutions required to transform one string into another. Jaccard distance [7] is derived from the *Jaccard index*, which is a generic measure of set similarity [4]. The Jaccard index $J(A, B)$ of two sets is the size of the intersection of the two sets divided by the size of their union. This yields a real number between 0 and 1; the Jaccard distance $d_J(A, B)$ is then $1 - J(A, B)$. Jaccard distance may be used as a string distance metric by taking the two sets A and B to be the sets of bigrams from the two strings under consideration.

These two distance metrics were used to compare potential matches from the BK-tree against an OCR result. Matches were ordered first in ascending order of edit distance from the OCR result, with Jaccard distance used to distinguish matches with the same edit distance. We define the sorting function $d_A(s, t_1, t_2)$, which returns 1 if t_1 is considered closer to s than t_2 and 0 if not.

2.2.2 Exact matching

Searching for an exact match was a relatively straightforward procedure, shown in Alg. 2. We compared the current word in the OCR results with the words surrounding it in the truth text, rather than simply comparing with the current word. In this way we could account for slight discrepancies in reading order between the OCR and the truth text. The function returns the index of the truth text word with the shortest metric distance from the OCR result, and a binary value indicating whether the match was exact or not.

Algorithm 2 Exact matching

```
 1: function ExactMatch($(i, j)$)
 2:     $k^* \leftarrow 0$
 3:     for $k \leftarrow 0, 1, -1, 2, -2$ do
 4:         if $O_i = T_{j+k}$ then
 5:             return $(j + k, 1)$
 6:         else if $d_A(O_i, T_{j+k}, T_{j+k^*}) = 1$ then
 7:             $k^* \leftarrow k$
 8:         end if
 9:     end for
10:     return $(j + k^*, 0)$
11: end function
```

2.2.3 Approximate matching

If no exact match could be found, we searched the entire truth text for the best approximate match to the OCR result using BK(T), following Alg. 3. The truth text string with the shortest metric distance from the OCR result is found, and we then find the index of the occurrence of that word with the smallest difference from our current position in the truth text to return.

A BK-tree is a tree data structure that organises its nodes according to their distance from each other according to some distance metric; in our case, nodes are ordered according to their edit distance from the first word in the truth text. This BK-tree can be used to find all the words in the truth text that are within a given edit distance of a query word in $O(\log n)$ time in the average case.

2.2.4 Special cases

The approach outlined above, while generally accurate, required a few specific modifications to cope with situations

Algorithm 3 Approximate matching

```
1: function ApproximateMatch($(i, j)$)
2:     for $k \leftarrow 1, 2, \ldots$ do
3:         $ms \leftarrow \{m : m \in \mathrm{BK}(T),\ d_L(O_i, m) \leq k\}$
4:         if $ms$ is not empty then
5:             sort $ms$ according to $d_A$
6:             return index of $ms_0$ closest to $j$
7:         end if
8:     end for
9: end function
```

where the OCR results contained word breaks not present in the truth text. Without special consideration, these additional word breaks would cause the OCR correction to introduce new errors into the document, even if the two halves of the split word were in themselves correct.

The first cause of such word breaks was the hyphenation of long words. This was solved by checking the last character of an OCR result prior to matching; if the last character was a hyphen, the word was combined with the one following it before searching for a correction, and the match was then divided into two fragments again to correct the OCR.

The second cause was the result of irregular letter spacing in the page image, causing the OCR engine to insert additional word breaks. To solve this, an additional step was inserted into the algorithm after the first check for an exact match. If exact matching failed, the OCR result was combined with the word following it, and a second search for an exact match was run using the new word. If a match was found, the first of the OCR results was expanded to contain the entire word, and the second of the results was deleted.

2.3 Insertion of hidden text layer

The final phase was to take the corrected OCR results and use them to construct a new PDF-IT document from the original page images.

Each separate OCR result was used to construct a PDF text object, using the provided bounding box coordinates to position the text on the appropriate page correctly. For each distinct font in the OCR results, a font object was inserted into the PDF document, followed by a series of text streams, each consisting of all the text objects present on a single page. The page objects themselves were then updated to refer to these new objects, and the document's cross-reference table was updated.

3. PERFORMANCE

In this section we demonstrate the effectiveness of the algorithm described in Sec. 2, by presenting the results of applying the OCR correction algorithm to four PDF documents. Truth texts for the documents were created by manually transcribing the documents into plain text files, preserving capitalisation, punctuation, and spelling but omitting text found in the header or footer of the document, such as page numbers.

Performance was measured simply by counting the number of errors present in the OCR results before and after correction, and calculating the percentage change in the number of errors. An *error* was defined as any word appearing in an incorrect place in the OCR results, relative to the original PDF document, or any word that was found to be missing in the results. The error rate was then the number of errors

Table 1: No. of errors in OCR results (all text)

Document	Errors		Change
	raw	corrected	
1	47	49	+4.26%
2	30	24	−20.0%
3	24	14	−41.7%
4	20	25	+25.0%

Table 2: No. of errors in OCR results (body only)

Document	Errors		Change
	raw	corrected	
1	44	15	−65.9%
2	30	23	−23.3%
3	24	13	−45.8%
4	17	9	−47.1%

in the OCR results divided by the total number of words in the results.

3.1 Results

Tables 1 and 2 show the number of errors found in the OCR results of four PDF documents before and after running the correction algorithm.

All four documents contained text that was not properly part of the document itself: page numbers, headers and footers, handwritten annotations, and other publisher artifacts. As noted above, such text was excluded from the prepared truth texts, on the grounds that such text would not be found in a truth text obtained, say, from an author-submitted typescript. This caused problems, because the OCR would generally recognise this extra text well (with the exception of handwriting), while the truth text would attempt to 'correct' what it saw as invalid text.

For completeness' sake, we present both sets of results, with Table 1 showing error count with artifacts included, and Table 2 showing error count with artifacts excluded.

3.2 Discussion

When publisher artifacts were excluded from the error count, all four documents saw a significant reduction in the number of errors in the OCR results. An inspection of the corrected OCR results indicated that the majority of errors that remained were the result of words being incorrectly split into *several* fragments, so that a string appearing as a single word to a human was interpreted as three or more separate words by the OCR. In Sec. 2.2.4 we described a method of identifying where a word had been split into two, but we do not attempt to check further for other breaks. In principle we could extend the solution outlined above to test three-word or even four-word strings, but it would be difficult to determine when to stop trying to combine words and instead to search for an approximate match.

4. CONCLUSIONS

By using a ground truth text containing a canonical version of the words found in the document, we were able to significantly improve the accuracy of the hidden text layer within a PDF-IT document.

Our method successfully matches the reading order of the OCR results to the truth text, but is still prone to being misled. Erroneous word breaks inserted by the OCR engine can cause the algorithm to attempt to 'correct' words that do not actually exist within the document, and such correction will alter the algorithm's perception of its position within the truth text. Despite this, once a well-recognised word is found, we are able to recover the correct position in the text. Our test results show that it is possible to align the truth text to the page image with a high degree of accuracy without a large amount of preprocessing of the truth text.

4.1 Future work

We feel very pleased at the success of this preliminary work but it is clear that some degree of pre-processing, using document recognition techniques, for recognising headers, footers and figures-with-text would certainly pay dividends. Equally there are certainly better OCR engines than Tesseract (the commercial alternatives such as Abbyy Fine Reader and Nuance Omnipage have a strong reputation)

A low-cost possibility to be investigated is whether retaining any Adobe-created OCR layer and comparing its recognition results with those of Tesseract might improve recognition performance, particularly if the bounding boxes of glyphs in the Adobe OCR stream could be extracted with a public domain tool such as PDF Box.

5. ACKNOWLEDGEMENTS

We thank Clive Adams of the Institute for Mental Health, University of Nottingham, for making available to us PDF documents from the Cochrane Schizophrenia Group's collection, around which this work is based.

6. REFERENCES

[1] S. Feng and R. Manmatha. A hierarchical, HMM-based automatic evaluation of OCR accuracy for a digital library of books. In *Proceedings of the 6th ACM/IEEE-CS Joint Conference on Digital Libraries, 2006. JCDL'06.*, pages 109–118. IEEE, 2006.

[2] J. D. Hobby. Matching document images with ground truth. *International Journal on Document Analysis and Recognition*, 1(1):52–61, 1998.

[3] J. Hughes, D. F. Brailsford, S. R. Bagley, and C. E. Adams. Generating summary documents for a variable-quality PDF document collection. In *Proceedings of the 2014 ACM Symposium on Document Engineering*, pages 49–52. ACM, 2014.

[4] P. Jaccard. The distribution of the flora in the alpine zone. *New Phytologist*, 11(2):37–50, 1912.

[5] T. Kanungo and R. M. Haralick. Automatic generation of character groundtruth for scanned documents: a closed-loop approach. In *Proceedings of the 13th International Conference on Pattern Recognition, 1996*, volume 3, pages 669–675. IEEE, 1996.

[6] D.-W. Kim and T. Kanungo. Attributed point matching for automatic groundtruth generation. *International Journal on Document Analysis and Recognition*, 5(1):47–66, 2002.

[7] M. Levandowsky and D. Winter. Distance between sets. *Nature*, 234(5323):34–35, 1971.

[8] I. Z. Yalniz and R. Manmatha. A fast alignment scheme for automatic OCR evaluation of books. In *2011 International Conference on Document Analysis and Recognition (ICDAR)*, pages 754–758. IEEE, 2011.