View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Nottingham ePrints

Pinkney, Alexander J. and Bagley, Steven R. and Brailsford, David F. (2011) Reflowable documents composed from pre-rendered atomic components. In: ACM Symposium on Document Engineering (DocEng '11), 19-22 Sept 2011, Mountain View, California, USA.

Access from the University of Nottingham repository:
http://eprints.nottingham.ac.uk/28128/1/eprint-reflow2011.pdf

Copyright and reuse:

The Nottingham ePrints service makes this work by researchers of the University of Nottingham available open access under the following conditions.

- Copyright and all moral rights to the version of the paper presented here belong to the individual author(s) and/or other copyright owners.
- To the extent reasonable and practicable the material made available in Nottingham ePrints has been checked for eligibility before being made available.
- Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way.
- Quotations or similar reproductions must be sufficiently acknowledged.

Please see our full end user licence at:
http://eprints.nottingham.ac.uk/end_user_agreement.pdf

A note on versions:

The version presented here may differ from the published version or from the version of record. If you wish to cite this item you are advised to consult the publisher's version. Please see the repository url above for details on accessing the published version and note that access may require a subscription.

For more information, please contact eprints@nottingham.ac.uk

Reflowable Documents Composed from Pre-rendered Atomic Components

Alexander J. Pinkney
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
azp@cs.nott.ac.uk

Steven R. Bagley
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
srb@cs.nott.ac.uk

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

ABSTRACT

Mobile eBook readers are now commonplace in today's society, but their document layout algorithms remain basic, largely due to constraints imposed by short battery life. At present, with any eBook file format not based on PDF, the layout of the document, as it appears to the end user, is at the mercy of hidden reformatting and reflow algorithms interacting with the screen parameters of the device on which the document is rendered. Very little control is provided to the publisher or author, beyond some basic formatting options.

This paper describes a method of producing well-typeset, scalable, document layouts by embedding several pre-rendered versions of a document within one file, thus enabling many computationally expensive steps (e.g. hyphenation and line-breaking) to be carried out at document compilation time, rather than at 'view time'. This system has the advantage that end users are not constrained to a single, arbitrarily chosen view of the document, nor are they subjected to reading a poorly typeset version rendered on the fly. Instead, the device can choose a layout appropriate to its screen size and the end user's choice of zoom level, and the author and publisher can have fine-grained control over all layouts.

Categories and Subject Descriptors

I.7.2 [**Document and Text Processing**]: Document Preparation—*format and notation, markup languages*; I.7.4 [**Document and Text Processing**]: Electronic Publishing

General Terms

Algorithms, Documentation, Experimentation

Keywords

PDF, COGs, eBooks, Document layout

1. INTRODUCTION

In recent years, the consumption of documents on mobile devices, such as eBook readers, has increased dramatically. However, the visual quality of a document on these devices is often lacking, when compared to other digital document systems (see figure 1). The result of an eBook reader's layout engine is often visually unappealing, with uneven spacing in consecutive lines of text, poor justification, and the lack of a sophisticated hyphenation system.

This is a far cry from the quality of typesetting available from PDF or PostScript documents. These vector-based, device-independent page description languages are able to create a digital version of the document that is identical in print. These page description languages, coupled with high-quality typesetting systems (such as TEX, troff or Adobe InDesign) have produced an expectation that digital documents will be of similar quality to that achievable through hand composition. TEX and Adobe InDesign, in particular, have excellent support for many of the subtle nuances used by hand compositors, which are often overlooked by more basic typesetting packages (e.g. automated support for kerning and ligatures). This quality does not come without a price: the algorithms used to calculate the layout are computationally expensive and so are run only once, to produce a PDF with a fixed layout targeted at a fixed page size.

EBook readers, it seems, have had to take a step backwards to simpler (and, therefore, less computationally expensive) algorithms to maximise the battery life of the device. The result is that the high-end hyphenation, kerning, and ligature support has had to be sacrificed and the on-screen result is reminiscent of the output of an HTML rendering engine or a very basic word processor.

This paper investigates an alternative approach to generating the display for an eBook reader. Here, the text is pre-rendered (using a high-quality typesetting algorithm) in several column widths, prior to display, when the document is created. At view time, the most appropriate column width is selected for display, the system balancing between excessive white space and multiple columns. Section 2 examines the problems posed by current eBook readers in further detail, while section 3 presents our initial prototype solution to some of these problems.

2. PROBLEMS WITH CURRENT EBOOK READERS

Three formats currently dominate the eBook market: EPUB and Mobipocket, which allow the document to be formatted to fit the device, and PDF, which does not. (PDF and EPUB are open standards; Mobipocket is the format upon which Amazon's Kindle format is based.) Both the EPUB and Mobipocket formats are largely based on XHTML. Whilst the use of an XML-derived format allows the semantic structure of documents to be very well defined, in general their presentation can only be specified in a very loose

FINAL DRAFT of Short Paper accepted for
ACM *DocEng'11,* September 19–22, 2011, Mountain View, California, USA.
Copyright 2011 Pinkney, Bagley and Brailsford.

campaign in which one candidate is a sure winner and you would like to bask in reflected glory or receive some future in-kind consideration. The one candidate you *won't* contribute to is a sure loser. (Just ask any presidential hopeful who bombs in Iowa and New Hampshire.) So front-runners and incumbents raise a lot more money than long shots. And what about spending that money? Incumbents and front-runners obviously have more cash, but they only spend a lot of it when they stand a legitimate chance of losing; otherwise, why dip into a war chest that might

5% Locations 343-49 6578

Figure 1: The Kindle 3 appears to primarily use justified text, falling back to ragged-right when inter-word spacing would become too large.

manner. The user is often presented with a choice of typefaces and point sizes, allowing the reader software to render the document in essentially any arbitrary way it chooses.

Conversely, PDF is entirely presentation-oriented, stemming from its origins as essentially 'compiled PostScript'. PDF, therefore, will often include no information on the semantic structure of the document, and will consist simply of drawing operators which describe the document pages. There is no compulsion for these drawing operators to render the page in an order that might be considered sensible: for example, if a PDF generator program decided to render every character on a page in alphabetical order, or radially outwards from the centre, the resulting file would still be semantically valid, and the result might well be unnoticeable to the end user. This lack of imposed semantic structure can make it difficult to infer the best way to 'unpick' PDF files to allow their content to be reflowed into a new layout.

Since an XHTML-derived format has no fixed presentation associated with it, this must be calculated each time the document is displayed, in a similar manner to the way an interpreted programming language needs to be interpreted each time it runs. For an eBook reader to maximise its battery life (the human reader will be annoyed if the device dies just before the climax of a novel!), the 'interpretation' needs to be as simple as possible — i.e. the algorithm used must not be too complex, since the more CPU cycles spent executing it, the less time the CPU can spend idle, and hence the greater the drain on the battery. Furthermore, the longer that is spent formatting the output, the longer the delay between page turns on the device, and with the speed of CPUs used in these devices ($<$ 500 MHz) it does not take too large an increase in computation for the page turn to become noticeable.

2.1 Hyphenation and Line-Breaking

EBook readers typically use a 'greedy' algorithm to lay out their text — that is, they place as many words as will fit onto the current line without exceeding it, then start a new line and continue. Although this algorithm is optimal in that it will always fit text onto the fewest possible lines, it often causes consecutive lines to have wildly varying lengths, accentuating either the 'ragged-right' effect of the text, or, in the case of justified text, the inter-word spacing. In general, eBook readers will only hyphenate in extreme cases — indeed the Kindle 3 seems not to do so at all. Knuth and Plass[7] developed a more advanced line-breaking algorithm (now used by TeX) which attempts to minimise large discrepancies between consecutive lines by considering each paragraph as a whole. TeX also uses the hyphenation algorithm designed by Liang[8], which has been ported to many other applications.

To AV V. Wa fi fl

To AV V. Wa fi fl

Figure 2: Examples of various letter-pairs and their kerned (left) or ligature (right) equivalents, as typeset by TeX.

2.2 Other Typographical Techniques

Other techniques employed during hand-typesetting and high-quality electronic typesetting include the use of kerning and of ligatures. Kerning involves altering the spacing between certain glyph pairs in order to produce more consistent letter spacing, whilst ligatures are single-glyph replacements for two or more single glyphs which may otherwise have clashing components. Some examples of these are shown in figure 2. Kerning requires a table of kern-pairs, specific to each font; values from this table must then be looked up for every pair of adjacent glyphs in the document. Ligatures may or may not need to be inserted: if the component characters of the ligature lie over a potential hyphenation point, it cannot be decided whether to replace them with the ligature until it is known whether the hyphenation point needs to be used.

3. A GALLEY-BASED APPROACH

Our proposed solution, of precomputing several text variants, revisits an approach to typesetting from before the advent of desktop publishing. In the days before DTP, newspaper articles were typeset into long columns known as *galleys*. Since all columns in the newspaper would be of uniform width, all articles could be typeset into galleys of the same measure, and then broken as necessary between lines, in order to slot into the final layout of the newspaper. Once the text has been set in this manner, with appropriate hyphenation and justification, the individual lines can be treated as atomic units and will never have to be re-typeset. In essence, each article is 'compiled' only once, but can be used anywhere in the final layout without penalty.

It is this behaviour we wish to emulate. So long as the atomic components of the document are tightly specified, and the reader software can obey the associated drawing instructions (essentially treating them as pre-typeset blocks), the resulting display of the document will be of as high typographic quality as that of the original galley, and the requirement for further computation will be vastly reduced. In order to permit aesthetic layout for a wider range of screen sizes, it seems sensible to create a document containing multiple renderings of the same content, and simply choosing the 'best fit' rendering when the document is displayed.

3.1 A Sample Implementation

Our sample implementation is built around our existing work in PDF and Component-Object Graphics (COGs)[1], but there is no reason why it could not be implemented in any other format capable of tightly specifying page imaging operations. It builds on existing software, principally *pdfdit*, in conjunction with *COG Manipulator*, as these tools are already capable of producing modular documents with tightly specified rendering.

3.1.1 The COG Model

The Component Object Graphic (COG) model was developed to enable the reuse of semantic components within PDF documents, by breaking the traditional graphically-monolithic PDF page into a series of distinct, encapsulated graphical blocks, termed COGs. In its original incarnation, the COG model did not account for any

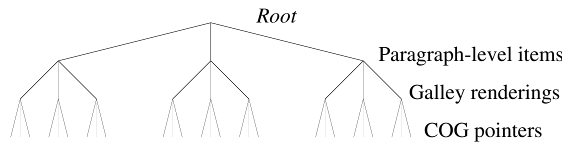

Figure 3: A simple document structure tree. The first level below the root represents all paragraph-level items: headings, paragraphs, figures etc. These items have one child for each galley rendering of the document. These in turn have one child for each COG comprising their content — in the case of a paragraph or heading: its lines; in the case of a figure: the figure itself and any associated caption.

relationship between individual COGs — it was simply designed as a method with which document components could be easily reused or reordered. The COGs it generates are largely at the granularity of a paragraph, and can still be imaged onto the page in any arbitrary order, independent of reading order.

In order to implement our galley-based design, it is necessary to decrease this granularity, such that each line of text is represented by a separate COG. However, it is also important that the semantic structure of the document is explicitly stored. This is principally so that the reading order of the COGs is maintained, and also so that the reader software can identify paragraphs, headings etc. to enable them to be laid out correctly.

The COG model takes advantage of the fact that the PDF specification allows the content of a page to be described by an array of streams of imaging operators, rather than the more commonly encountered monolithic stream. Unfortunately, this array can only be one-dimensional, meaning that while it can enforce the reading order, it cannot be used to, say, group lines into paragraphs. Since the PDF specification allows essentially arbitrary insertion of data structures into a document (PDF readers which do not recognise these will simply ignore them), this flexibility was used to embed a simple tree structure representing the paragraphs, in parallel to the COGs themselves (an example of which is shown in figure 3). At the level of its leaves, this tree simply contains pointers to the COGs which make up the content of the document. In the simplest case, where the document contains only one rendering (and thus the paragraph-level items have only one child) the COGs pointed at by the leaves can simply be rendered in order, adding vertical space as appropriate.

3.1.2 The Source Document

Since the majority of available tools for producing COGged PDFs rely on the typesetting package *ditroff*, it was decided to use this as the basis for the source document. Ditroff is particularly amenable to many of the features required here — it is quite happy to have its page length set to large numbers — one sample document used a page length of 2000 inches (approximately 50 metres) with no complaints from ditroff. The line length was set to a small value (approximately two inches) in order to produce a narrow column of text. Following this, the actual document content was inserted several times, and the line length incremented, producing one document effectively containing multiple galley renderings of the same content.

3.1.3 pdfdit

Having generated the source document, it was processed with ditroff to generate the intermediate code used to feed each typesetter post-processor. This output is very expressive, and, unlike

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras vel enim vitae mauris vestibulum egestas. Suspendisse potenti. Pellentesque leo nunc, lobortis vitae gravida vel, congue at nulla. Praesent a placerat mauris. Praesent sed erat ac dui tincidunt consectetur vel nec leo. In velit odio, congue non eleifend at, accumsan eu diam. Suspendisse dignissim, quam quis euismod laoreet, est leo euismod lectus, sed consequat leo nunc in ante. Duis risus tellus, suscipit ut fermentum et, ornare non lorem. Morbi nibh elit, dignissim ullamcorper posuere at, lacinia condimentum odio. Fusce vitae metus mi. Pellentesque scelerisque fermentum magna a dictum. Mauris ut ante mauris, ac viverra felis. Praesent ut elit ut purus malesuada suscipit. Fusce mollis eros ac lectus suscipit gravida. Pellentesque vel nisl nec eros convallis luctus nec eu quam. Aliquam tincidunt ultrices blandit.

Vestibulum lorem felis, consectetur ornare vehicula ac, cursus tincidunt nisl. Aliquam in

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras vel enim vitae mauris vestibulum egestas. Suspendisse potenti. Pellentesque leo nunc, lobortis vitae gravida vel, congue at nulla. Praesent a placerat mauris. Praesent sed erat ac dui tincidunt consectetur vel nec leo. In velit odio, congue non eleifend at, accumsan eu diam. Suspendisse dignissim, quam quis euismod laoreet, est leo euismod lectus, sed consequat leo nunc in ante. Duis risus tellus, suscipit ut fermentum et, ornare non lorem. Morbi nibh elit, dignissim ullamcorper posuere at, lacinia condimentum odio. Fusce vitae metus mi. Pellentesque scelerisque fermentum magna a dictum. Mauris ut ante mauris, ac viverra felis. Praesent ut elit ut purus malesuada suscipit. Fusce mollis eros ac lectus suscipit gravida. Pellentesque vel nisl nec eros convallis luctus nec eu quam. Aliquam tincidunt ultrices blandit.

Vestibulum lorem felis, consectetur ornare vehicula ac, cursus tincidunt nisl. Aliquam in enim nisi, quis hendrerit est. Nullam pretium congue sapien ac tincidunt. Suspendisse suscipit felis eget nibh luctus sit amet imperdiet ligula venenatis. Vestibulum eu dui nulla. Vivamus interdum ullamcorper sapien eget dapibus. Proin sed dictum

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras vel enim vitae mauris vestibulum egestas. Suspendisse potenti. Pellentesque leo nunc, lobortis vitae gravida vel, congue at nulla. Praesent a placerat mauris. Praesent sed erat ac dui tincidunt consectetur vel nec leo. In velit odio, congue non eleifend at, accumsan eu diam. Suspendisse dignissim, quam quis euismod laoreet, est leo euismod lectus, sed consequat leo nunc in ante. Duis risus tellus, suscipit ut fermentum et, ornare non lorem. Morbi nibh elit, dignissim ullamcorper posuere at, lacinia condimentum odio. Fusce vitae metus mi. Pellentesque scelerisque fermentum magna a dictum. Mauris ut ante mauris, ac viverra felis. Praesent ut elit ut purus malesuada suscipit. Fusce mollis eros ac lectus suscipit gravida. Pellentesque vel nisl nec eros convallis luctus nec eu quam. Aliquam tincidunt ultrices blandit.

Vestibulum lorem felis, consectetur ornare vehicula ac, cursus tincidunt nisl. Aliquam in enim nisi, quis hendrerit est. Nullam pretium congue sapien ac tincidunt. Suspendisse suscipit felis eget nibh luctus sit amet imperdiet ligula venenatis. Vestibulum eu dui nulla. Vivamus interdum ullamcorper sapien eget dapibus. Proin sed dictum arcu. Curabitur velit justo, fringilla non sodales laoreet, dignissim nec nulla. Praesent convallis

Figure 4: Sample renderings from the Acrobat plugin at page widths of 42, 48, and 54 em.

TEX's DVI, contains enough information that post-processors are easily able to locate the start and end of lines and paragraphs within the document. This meant that only minimal changes were needed to be made to the *pdfdit* package described in [1] to implement our design.

The first change necessary was to decrease the granularity of the output COGs, producing them at the line level, rather than at the paragraph level. Secondly, some method of generating the requisite tree representing the document structure was required. This was solved by simply using the point at which the original version of pdfdit would have started a new paragraph-level COG, and, instead, starting a new paragraph-level block entry in the document structure tree. Each subsequent line-level COG produced can then be added as a child of this block.

Once the entire output file has been parsed, the tree representations of the various width galleys are amalgamated per-paragraph, as indicated in figure 3, and finally the PDF file is serialised, replete with COGs and content tree.

3.1.4 Acrobat Plugin

The decision to use Acrobat as an eBook 'emulator' stemmed once again from the available existing COG-based tools, as well as the extensive API and developer support available for Acrobat. Moving a COG on a PDF page is as simple as deleting its associated spacer object from the content array of the page, creating a new spacer containing the COG's desired new position, and then adding that back to the content array.

Since, by this point, most of the computationally expensive typesetting has already been carried out, the algorithm used to lay out the lines of the galleys can be very simple. The plugin chooses the most appropriate galley width to lay out, based on the current page width, and according to some measure of aesthetic, and then simply lays the document out line by line, with appropriate vertical spacing, until no more lines will fit in the current column. Any subsequent columns which will fit on the same page are then laid out in the same manner.

3.1.5 Layout and Metrics

Since galleys of text lend themselves to being used in a columnar format, a method of fitting columns appropriately to the available page width must be devised. A sensible first approach is simply to calculate how many columns of each galley rendering will fit, by adding the galley width to a specified minimum inter-column

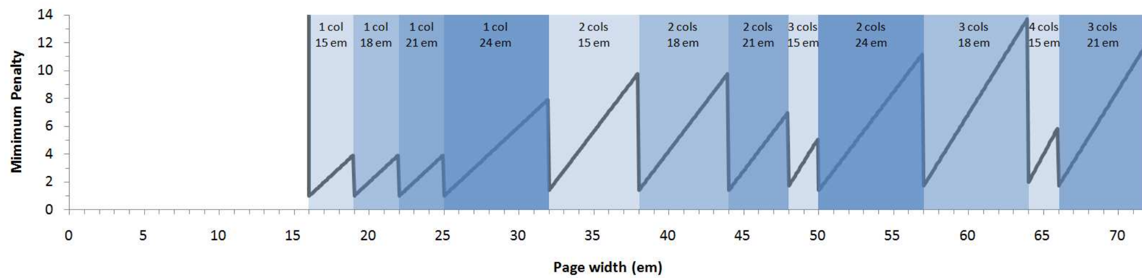

Figure 5: Graph showing the minimum penalty value of all galleys in a reflowable document, over a range of page widths. The particular document used contained four galleys; these were rendered at widths of 15, 18, 21 and 24 em, with a minimum gutter width of 1 em. Each vertical band highlights a range of page widths within which only the horizontal spacing of the page is altered. The boundaries between vertical bands represent a switch between galley renderings — the galley used and number of columns is as annotated on the graph.

spacing, and dividing the page width by this. The remainder of this division will then specify the total extra amount of horizontal whitespace required, which can then divided up and inserted between the columns. A simple measure of aesthetic here is to apply a linear penalty for any extra whitespace required, as we seek to keep page margins and column gutters to a minimum.

As the page width increases, so must the widths of the inter-column gutters. In accordance with the extra-whitespace penalty, each galley rendering will produce penalties which vary in a saw-tooth manner as the width of the page is increased. With a careful choice of galley widths, when these sawtooth penalties are overlaid, and the galley producing the minimum penalty chosen at each page width, a flatter and finer-toothed penalty-graph emerges, as shown in figure 5.

In addition to penalising extra whitespace, wider columns should, in general, be favoured over narrower ones, i.e. for a given page width, fewer, wider columns are generally considered preferable to a greater number of narrower columns. By multiplying the existing penalty by a smaller-than-linear function of the number of columns (experiments have been carried out with both logarithms and roots) the penalty may be subtly increased for greater numbers of columns. The formula for the penalty used in figure 5 is $P = (C + W_{ex}) \cdot \sqrt{N_{cols}}$, where P is the penalty, W_{ex} is the extra whitespace required to be inserted, N_{cols} is the number of columns which are required to fill the width of the page, and C is a positive constant. The purpose of the constant is to prevent the penalty from ever evaluating to zero, which would have the effect of disregarding the weighting of the number of columns. Figure 5 uses $C = 1$.

4. CONCLUSIONS AND FUTURE WORK

This paper outlines our initial exploration of the idea of using pre-rendered galleys for eBooks. So far, our initial implementation has generated multicolumn layouts that look acceptable, and we believe there is mileage in continuing to investigate this method. However, there is still a lot of work to be done. Firstly, a very simple formula is used to determine which column width variant to select, and we are investigating the suitability of other methods of determining aesthetically pleasing layouts (such as those outlined in [2, 3, 4, 5, 6, 9]). Also, our system does not currently allow the font size to be changed (since it is fixed when the galleys are created). One approach to allow the font size to be changed would be to scale smaller column width variants up to larger columns. For example, if the 15 em wide variant is scaled up to 18 em, then text would be scaled up by 20% — the equivalent of converting 10 pt text to 12 pt.

It should also be noted that optimal placement of floating blocks cannot be 'compiled out' in the same manner that hyphenation and line breaking can; these will still need to be positioned into the relevant places as the document is displayed. If the simple approach is taken that floats should be placed at the top of a column or after another float, a document layout somewhat reminiscent of this one will emerge, although the floats will inevitably tend to drift towards the end of the document, away from their desired position.

Finally, to confirm that this method has validity it needs to be implemented in an actual eBook system, rather than simulated in Acrobat. There, it will be possible to compare the performance of our system with both a normal eBook renderer, and one that has been enhanced to use a sophisticated hyphenation and justification algorithm.

5. REFERENCES

[1] S. R. Bagley, D. F. Brailsford, and M. R. B. Hardy. Creating reusable well-structured PDF as a sequence of component object graphic (COG) elements. In *Proceedings of the 2003 ACM Symposium on Document Engineering*, pages 58–67. ACM Press, 2003.

[2] H. Y. Balinsky, J. R. Howes, and A. J. Wiley. Aesthetically-driven layout engine. In *Proceedings of the 2009 ACM Symposium on Document Engineering*, pages 119–122, 2009.

[3] R. Bringhurst. *The Elements of Typographic Style (v 3.2)*. Hartley & Marks, 2008.

[4] E. Goldenberg. Automatic layout of variable-content print data. Master's thesis, University of Sussex, 2002.

[5] S. J. Harrington, J. F. Naveda, R. P. Jones, P. Roetling, and N. Thakkar. Aesthetic measures for automated document layout. In *Proceedings of the 2004 ACM Symposium on Document Engineering*, pages 109–111. ACM Press, 2004.

[6] R. Johari, J. Marks, A. Partovi, and S. Shieber. Automatic yellow-pages pagination and layout. Technical report, Mitsubish Electric Research Laboratories, 1996.

[7] D. E. Knuth and M. F. Plass. Breaking paragraphs into lines. *Software — Practice and Experience*, 11:1119–1184, 1981.

[8] F. M. Liang. *Word Hy-phen-a-tion by a Com-put-er*. PhD thesis, Stanford University, 1983.

[9] L. Purvis, S. Harrington, B. O'Sullivan, and E. C. Freuder. Creating personalized documents: an optimization approach. In *Proceedings of the 2003 ACM Symposium on Document Engineering*, pages 68–77. ACM Press, 2003.