Review of Automatic Document Formatting

Nathan Hurst
Adobe Systems Inc.
345 Park Ave.
San Jose, CA 95110
hurst@adobe.com

Wilmot Li
Adobe Systems Inc.
345 Park Ave.
San Jose, CA 95110
wilmotli@adobe.com

Kim Marriott
Clayton School of IT
Monash University
Vic. 3800, Australia
Kim.Marriott@infotech.monash.edu.au

ABSTRACT

We review the literature on automatic document formatting with an emphasis on recent work in the field. One common way to frame document formatting is as a *constrained optimization problem* where decision variables encode element placement, constraints enforce required geometric relationships, and the objective function measures layout quality. We present existing research using this framework, describing the kind of optimization problem being solved and the basic optimization techniques used to solve it. Our review focuses on the formatting of primarily textual documents, including both micro- and macro-typographic concerns. We also cover techniques for automatic table layout. Related problems such as widget and diagram layout, as well as temporal layout issues that arise in multimedia documents are outside the scope of this review.

Categories and Subject Descriptors

I.7.2 [**Document and Text Processing**]: Document Preparation—*Format and notation, Photocomposition/typesetting*

General Terms

Algorithms

Keywords

adaptive layout, optimization techniques, typography

1. INTRODUCTION

Automatic document formatting underlies many widely used document processing applications. While early document authoring tools in the 60s and 70s provided relatively little automatic formatting due to limited computing power, more recent document authoring tools including TEX [49], Adobe® InDesign®, QuarkXPress®, and Microsoft® Publisher provide semi-automatic layout to relieve the burden on the author.

DocEng'09, September 16–18, 2009, Munich, Germany.

In the last fifteen years there has been a resurgence of interest in automatic layout because of the World Wide Web (WWW) and variable data printing (VDP). This has resulted in a shift of focus from micro-typographic concerns such as line breaking to macro-typographic concerns such as page layout. One of the design goals of modern web document standards such as HTML and CSS has been to separate the document content from its presentation so as to allow the layout to adapt to different viewing devices and to different user requirements, such as for larger fonts. Furthermore, dynamic content makes it impossible for the author to fully specify the final layout of a document. This is an issue for web pages and also for VDP in which improvements in printer technology now allow companies to cheaply print material which is customized to a particular recipient.

We believe the need for high-quality automatic layout will only increase. The devices used to view documents are becoming more varied (print, monitor, eBook, PDA, etc.) making automatic layout adaptation increasingly desirable, and rapidly growing information about user desires and requirements allows better customization of document content. Thus, developing better high-quality automatic layout should continue to be a central concern of the document processing community over the next decade.

Hundreds of articles have been written on automatic document formatting. They have appeared in a variety of venues and many of them are relatively recent. Since the last surveys of automatic formatting are more than fifteen years old [25, 24], we felt that it would be useful and timely to review this literature. We focus on recent papers, identify promising approaches and also problem areas. One difficulty with a review of this kind is delimiting what is meant by document formatting. We focus on primarily textual documents and do not consider widget layout or any kind of diagram layout. However, we do include table layout because this problem can be regarded as a restricted type of page layout with similar global tradeoffs. We also do not consider temporal layout issues that arise in multimedia documents or cropping/resizing of images.

High-quality automatic document formatting is an extremely difficult problem. The difficulties are threefold. First, it is hard to quantify what makes a good layout. While it is unrealistic to expect automatic layout systems to rival the creativity of a good graphic designer, we do need precise methods to identify when a layout is obviously wrong so as to be able to generate a layout which is reasonable. Second, layout is computationally difficult. Even seemingly simple sub-tasks like table layout or certain kinds of float place-

ment are NP-Hard. The third difficulty is the complexity of designing and implementing layout tools. This is because it is difficult to cleanly break layout into well-defined subtasks since decisions interact – line breaking can affect figure placement – and typographic conventions are extremely varied. Thus, it is easy to end up with a complex, ad hoc and unmaintainable software system.

We believe that a good way of approaching document formatting is as a *constrained optimization problem*. Decision variables encode where to place the elements, constraints enforce required geometric relationships, such as alignment or containment in a page, and the objective function measures the quality of the layout. Viewing automatic layout as a kind of constrained optimization problem provides some help with the above three issues. It encourages separation of the problem specification from the techniques used to solve it and forces a precise statement of the problem and objective function. It also encourages the exploration of different ways to solve the problem and the use of generic techniques for constrained optimization which can simplify software development. Our review will be based on this perspective.

A contribution of the review is to present existing research in terms of the kind of optimization problem being solved and the basic optimization technique used to solve it. We provide a brief introduction to constrained optimization in Section 2. The other main dimension we use to classify the research is the conceptual stage in document formatting. In Section 3 we start with micro-typographic concerns: kerning, line breaking and justification. In Section 4 we cover macro-typography: choice of the *logical layout* which specifies the elements and their abstract layout on each page, and *fine tuning* of the layout to adapt to dynamic content and the viewing environment. We treat table layout as a separate issue because of the many specialized algorithms that have been investigated. These are presented in Section 5.

2. CONSTRAINED OPTIMIZATION

In this section we briefly review the main generic approaches to solving constrained optimization problems that have been considered in document formatting.

It is standard to distinguish between *continuous problems* in which the variables range over the real numbers, *discrete problems* in which the variables can range over discrete choices and *mixed problems* in which variables can be either continuous or discrete. Both continuous and discrete optimization techniques have been used for document formatting.

2.1 Continuous problems

Two common approaches for solving continuous problems are variable elimination and iterative techniques [10, 67]. *Variable elimination* works by simplifying the problem to a solved-form in which each variable has a simple expression giving its value in terms of previously computed variables. Examples includes Gauss-Jordan elimination for solving systems of linear equations and the Simplex Algorithm for solving linear programming problems. Variable elimination approaches work well for simple constraints such as linear constraints but are not well-suited to complex non-linear constraints.

Iterative techniques, such as gradient descent, repeatedly improve the current solution. Here the main difficulty is ensuring that they converge in reasonable time and that they converge to a global minimum rather than just a local minimum. Interior point methods are a kind of iterative technique often used to solve *convex* problems (such as linear, quadratic and conic programming problems) in which any local minimum is guaranteed to be a global minimum.

One simple class of optimization problems are one-dimensional minimization problems. Here, binary or secant-like iterative search methods provide a simple and efficient solving technique. As we shall see such problems occur reasonably often in document layout.

2.2 Discrete problems

There are two main approaches to solving discrete constrained optimization problems: constructive search [73] and local search [26]. Both can use stochastic techniques to escape from local minimum although this is more common with local search.

Constructive search methods incrementally construct the layout by extending a partial solution, exploring different choices in the extension. In the case of document layout, for instance, they might choose at each step how to extend the current layout by adding a line of text or adding a figure. Efficiency requires that state which are equivalent with respect to future construction steps but which have been reached by different construction steps are merged into a single state – this is the key idea behind dynamic programming. Heuristics can be used to guide the search and to prune the search space. If possible, safe heuristics which guarantee that an optimal solution will be found are preferred, but unsafe heuristics are often used for efficiency.

The $A^\star$ class of algorithms are a good example of a safe constructive search method. For each state they use a conservative estimate of the penalty of the best complete solution that could be obtained by extending the state. At each step the state with the lowest estimate is chosen for expansion.

Somewhat related are branch-and-bound based mathematical programming techniques in which a conservative continuous relaxation of the problem is used to guide the search. This is a standard approach for solving Mixed Integer Linear Programming (MILP) problems. Interestingly, MILP techniques are very rarely used in document layout. In text and document layout a useful continuous approximation has been to view text as an incompressible liquid and approximate it using a non-linear area constraint or conjunction of linear inequalities.

Beam search based methods in which only the k-best partial solutions (based on some heuristic penalty function) are considered at each step are an example of an unsafe heuristic approach which is not guaranteed to find the optimal solution. An extreme of beam search are greedy techniques in which only the best partial solution is chosen at each step.

Local search methods[1] work with complete solutions. They include trajectory methods such as simulated annealing which try to improve a single solution by exploring the neighbouring solutions and population based meta-heuristic techniques like genetic algorithms. These are very powerful generic techniques. Efficiency crucially relies on good search heuristics and definition of the "neighbourhood." Limitations are lack of predictability – changing the problem slightly

[1]While we have used local search for this class of optimization methods there is as yet no widely accepted term – local search is sometimes used to refer to trajectory methods.

can lead to quite different solutions even if the technique is deterministic – and difficulty in handling constraints. Hill-climbing methods are a trajectory search in which the current solution is iteratively improved by moving to a better neighbour until there is no better neighbour, in which case a local optimum has been reached.

3. MICRO-TYPOGRAPHY

Micro-typography is concerned with the low-level composition of text: kerning, line breaking and justification. Automated techniques are increasingly used in practical applications.

3.1 Kerning

The first issue is how close to place adjacent characters. The obvious approach is to base this on the character's bounding box but this can lead to visually different spacing between letters. To overcome this optical kerning can be used in which the visual appearance of the characters is taken into account. Closely related is optical margin alignment in which the visual appearance of the characters is taken into account when placing characters on the margin of a text block so as to achieve a visually even margin. While good fonts provide kerning for each pair of characters in the font there is still a need for automatic kerning since: (a) many fonts do not provide good kerning information or do not give it for different point sizes and (b) margin kerning information for optical margin alignment assumes vertical margins and so does not handle layout with non-rectangular cut-outs etc., and (c) does not handle mixed font text, such as drop caps or a mixture of italic and roman fonts. There has been little published research on automatic optical kerning. The kf-module of the *hz* typesetting program provided the first automatic optical kerning that we know of [46, 81, 53, 32] and Adobe InDesign also provides automatic kerning.

3.2 Line breaking

The next optimization problem is to determine which word (or part of a word if hyphenation is allowed) to place at the end of the line. Historically this problem has been known as 'h and j', hyphenation and justification. Automatic line breaking algorithms use constructive search methods, adding a line of text at a time. The first document layout used a simple *first-fit* strategy in which words were added until no more could fit.[2] This an example of a greedy algorithm. TROFF [47] used a local heuristic algorithm to try and neaten up pairs of lines.

The seminal paper on line breaking is due to Knuth and Plass [50] who formulated line breaking and hyphenation as an optimization problem and gave an efficient dynamic programming algorithm to solve it that is used in the TEX document layout system. It works a paragraph at a time and tries to average out the length of the lines. If the line length is very short it will try and use hyphenation but this incurs a substantial penalty, especially if the previous line was hyphenated. Other packages that provide automatic line breaking include the *jp*-module from the *hz* typesetting program, Adobe InDesign, and the Breakers plug-in for QuarkXPress from Blue Sky TeX Systems.

While the Knuth-Plass algorithm works quite well it has a number of limitations [65]. First, the objective function is relatively unsophisticated and, for example, does not penalize "rivers" or handle ragged right text well. Second, it does not handle variable height text laid out in a non-rectangular region since it requires the length of each line to be known before computing the layout. This problem appears to be computationally much more difficult.

The Knuth-Plass algorithm has worst case complexity of $O(kn)$ where n is the number of words[3] and k the maximum number of words that can fit in a line. However, pruning reduces the effective complexity to linear time, as described by Hirschberg and Larmore [33] and Eppstein et al. [22]. In [40], the Knuth-Plass algorithm was extended to allow alternative wording of phrases including the use of abbreviations. Thus, another decision in the algorithm is the choice of wording. This mirrors the behaviour of human designers who will slightly rephrase text if it leads to better line breaking. We also mention extensions to the algorithm to line breaking of musical notation [31, 66].

3.3 Line justification

Once we have identified the line breaks the next step is to fine-tune the placement of characters and words on the line. This is typically only performed for fully justified text where the text must be laid out evenly to fill the entire line. Possible techniques are to use alternative ligatures and glyphs, scale the font, scale inter-word spacing and scale inter-character spacing.

Justification is naturally modelled as one-dimensional continuous optimization problem. TEX does this. The *box and glue* model was introduced in TEX. Lines are composed of boxes separated by glops of glue each of which has a default or natural width, a stretchability and a shrinkability. TEX computes an *adjustment ratio* that stretches (for short lines) or shrinks (for long lines) each glue glop by an amount proportional to the total on that line. So a glue glop that is twice as stretchable would get twice as much whitespace when extending a line [48]. While TEX only considers inter-word spacing, the basic approach can also be used to scale the font and inter-character spacing. The program *pdfTeX* does this [75]. The program *hz* and Adobe InDesign also can scale the font and inter-character spacing as part of line justification.

4. MACRO-TYPOGRAPHY

We now look at the macro-typographic aspects of automatic document layout. This covers the overall appearance of the layout and includes placement of objects such as floating figures, choice of content and number, size and placement of columns. Research on automatic layout in this area is quite varied. One reason for this variation is the kind of document being laid out and, in particular, how tightly its content is connected and ordered. For instance, a wiki page or conference paper has a highly connected *sequential* stream of components, while a poster or newspaper contains largely independent *non-sequential* components. Another is-

[2]Perhaps surprisingly, even today most word processors, web browsers and user interface toolkits still use the simpler first-fit line breaking algorithm in preference to optimal line breaking algorithms, for ease of implementation, minimising the cognitively disruptive rearrangement of text and perhaps efficiency.

[3]More exactly the number of unbreakable components in the text

sue is the time budget available to the layout system. Offline algorithms such as used for books and yellow pages directories can afford lengthy search times as their cost is amortized over large print runs. Mobile phone displays require low latency and low resource usage. However, perhaps the main reason for the variation in the approaches are different document and page layout models for logically structuring the document elements.

4.1 Document and page layout models

Different document layout models arise because documents are viewed on a variety of media. We have identified four basic models. The first is layout on a single fixed size page and is used for instance for a printed poster presentation. The second is layout on an unbounded number of fixed size pages and is typically used for print media and for PDF documents.[4] The third – *vertical scroll layout* – is layout on a single page of fixed width but unbounded height and is the standard model for viewing HTML and most web documents. The fourth model – *horizontal scroll layout* – is layout on a single page of fixed height but with unbounded width. While this model is less common we believe that it will prove to be the most popular model for multi-column layout on electronic media [16].

The page layout model specifies the kinds of spatial arrangements that the elements on the page can take. They include:

- *Coordinate:* This model is very simple and general – each object is given a location by specifying its coordinates on the page. This model is used in postscript, pdf and, with some extensions, in SVG.
- *Flow:* This simply places elements from a single stream consecutively on the page. It was the basic model used in the first versions of HTML.
- *Grid:* In this model objects on the page are organized using axis aligned grid lines whose position is a global property of the page template. The grid is a standard approach used by graphical designers and is common in magazine layout and presentation software such as Microsoft PowerPoint.
- *VH-Box:* This is a hierarchical model in which a box is either a simple object, a vertical stack of boxes or a horizontal sequence of boxes. This is used in TEX.
- *Guillotine:* This uses a structure known as an orthogonal space partition tree to represent the allocation of space on the page. Starting with a single page we can draw a line either horizontally or vertically across the whole page. In each of these two regions we can similarly split either horizontally or vertically.
- *Box:* The page is organized as a an arbitrary collection of rectangular regions. This generalizes the VH-Box, Guillotine and Grid models. Tables (Section 5) can also be regarded as a kind of Box layout.

Automatic macro-typography is conceptually split into two tasks: Choosing the logical page layout for the document (i.e., the elements and the basic geometric relationships between them in each page), and fine-tuning/adjusting the logical layout. We first discuss approaches that address each of these tasks and then describe some recent work on automatic layout re-targeting that considers aspects of both problems.

[4]Books are often printed in groups of 16 sides called signatures. Here, there is an opportunity to minimize unused paper in such groups.

4.2 Fine-tuning

The simpler problem is given a basic page layout how to fine-tune and adapt this to slightly different content, style or page size. While a wide variety of techniques have been used, almost all use continuous optimization techniques.

One-way constraints are the simplest and most common approach to fine-tuning a logical page layout. A one-way constraint is exactly like a formula in a spreadsheet cell. It has the form $x = f_x(y_1, ..., y_n)$ where the formula f_x details how to compute the value of the *output* variable x from the *input* variables $y_1, ..., y_n$. We say that the output variable *directly depends* upon its input variables. They are solved using a very simple kind of variable elimination.[5] One-way constraints allow the designer to specify the placement, size and style of the page elements as functions of the viewport dimensions, viewer specified style and the size of dynamic content. They have been used in, for example [61, 37, 41, 63, 59, 74]. The CSS formalism can also be viewed as being a restricted kind of one-way constraint based formalism in which the only constraints allowed are unary functions of the parent element attributes [15].

One-way constraints are simple to implement and can be solved extremely efficiently. They are also very versatile since f_x can be any function. However, they have two limitations which limits their usefulness.

The first limitation is that a variable can only be the output variable of at most one constraint and cyclic variable dependencies are not allowed.[6] An example of where a cyclic dependency arises in layout is vertical centering of some text in a non-rectangular shape such as an ellipse. This is inherently cyclic because the height of the text depends on how high up the layout starts in the ellipse as this affects the length of the text lines. Thus this cannot be computed using one-way constraints. The second limitation of one-way constraints is that they only provide a method for satisfying a set of constraints. They do not provide support for solving global optimization problems.

In part because of these limitations, hierarchical multi-way propagation constraints were used in [78, 79, 28] and linear arithmetic constraints were used in [12, 43, 3, 4, 58].

The main disadvantage of linear arithmetic constraints is that they cannot handle the non-linear constraint that a block must be large enough to contain its content. Here continuous approximation techniques have proven useful. Lin [55] used a dynamic linear approximation to the containment constraint followed by a clean-up pass fixing text widths and heights to handle such constraints. This created a powerful layout engine for adjusting a box layout to variable data. Hurst and Marriott [35] used a specialized iterative gradient projection technique to fine tune the placement of floating figures in a vertical scroll.

The other main technique that has been used for fine-

[5]All that the constraint solver needs to do is to order the constraints so as to ensure that input variables of each formula have been computed before the formula is evaluated to give the value of the output variable.

[6]In some systems cyclic variable dependencies are allowed but the solution found is not guaranteed to satisfy the constraints.

tuning layouts are one-dimensional optimization techniques. They are used in TEX to fine-tune HV-Box layouts by recursively spacing out glue glops in vertical and horizontal box distributions and Hurst [34] investigated their use to recursively fine-tune guillotine layouts. They are also used by Lin [56] to appropriately scale the font size to fit the textual content in a fixed size document.

Column balancing was provided in Type & Set [2] using a dynamic programming approach. Another approach is to use one-dimensional optimization techniques [39] – these can also be used for vertical centering. Type & Set also provided a dynamic programming algorithm to fine-tune allocation of text to pages to globally minimize the number of widows and orphans.

Finally, we mention the fine-tuning approach of Loureiro and Azevedo [57] who use MILP techniques to fine tune a layout to reduce the number of pages in commercial bill printing. This works for smaller documents but probably does not scale up to larger documents. As suggested in [39] one-dimensional optimization techniques seem like a more scalable approach.

4.3 Choosing a page layout

The highest-level decision in automatic document layout is the choice of elements on a page and their logical layout within the page. This is the most difficult sub-problem in automatic document layout. Since it is basically a discrete problem, constructive and local search are the most commonly used techniques.

Floating figure placement has attracted some attention. Here, the aim is to place floats close to references in text and usually to preserve the order of the floats. The most common approach, used in TEX and HTML, is a greedy approach in which each float is placed at the best available location when it is first encountered. An extension to TEX to handle float placement in multi-column layout has been proposed by Mittelbach [64]. This also utilizes a greedy heuristic.

A variety of complete constructive search methods for optimal float placement have also been investigated. The main problem that has been studied is float placement in paged documents where the floats are placed at the top and/or bottom of the page and are effectively as wide as the page. This is a kind of flow or HV-Box page layout. The objective of the layout is to ensure that each float occur on the same page as the first reference to it or on a following page as close as possible to the reference. The main decisions in the layout are on which page to place each float and how much white space to leave at the bottom of each page. Plass [69] investigated how the precise formulation of this problem affected its difficulty. He showed that if a quadratic penalty function is used then the problem is NP-hard, but that if a linear penalty function is used then the optimal layout can be computed in polynomial time using a dynamic programming algorithm. Brüggemann-Klein et al. [17] further investigated the use of dynamic programming for this problem.

Placing floats in multi-column layout was considered by Marriott et al. [62]. They investigated the use of dynamic programming and A* techniques. Related is yellow pages layout on multi-column pages. This was investigated by Johari et al. [42] who used simulated annealing. In their version of the problem there were two ordered sequences of content: floating ads which had to be placed at the bottom of each page and the alphabetical listing of businesses which were placed on top of the ads. The aim was to place ads close to their category and to reduce the number of pages. This problem was also addressed by Graf's YPPS [29] system. It was based on extending [28] and using a combination of multi-way propagation and finite domain label inference, became a commercial product. Hurst and Marriott [35] used local search techniques to place an unordered sequence of floating figures on a vertical scroll. They showed that this float placement problem was NP-Hard.

Feiner [23] introduces automatic layout using computed grids, the grid lines are computed from the page size and elements are fitted sequentially to the grid lines. Elements are uniformly scaled to their grid bounds, elements which do not fit are either moved to the next page or ignored.

Also related is the work on automatic choice of templates for grid-like layouts by Jacobs et al. [41]. They used a dynamic programming algorithm to choose a template for each page in which figures could occur at locations specified with one-way constraints. This also allows alternative choices of textual content. With multiple streams of text, such as found in newspapers and magazines we also have the problem of determining a packing for textual elements. Oliveira [19] provides two algorithms for packing continuous area text into rows and columns for a newspaper like layout. The use of templates for constructing newspaper layouts is well understood, with several techniques dating from 1970s discussed in Lie's master's thesis, *The Electronic Broadsheet* [54]. Lie describes his approach as 'playing tetris', but provides few details. He describes the work of de Treville [20] and Kan [45], who both provide general purpose geometric templates for describing and laying out newspapers automatically using linear constraints. Schrier et al. [74] reexamined this with a language for generating templates. This was the basis for the Times Reader[7] for reading on-line newspaper articles.

Automatic generation of guillotine page layouts has been considered by Goldenberg [27] who used genetic algorithm techniques similar to those used in VLSI layout. Genetic algorithms were also used in a system presented by Harrington/Purvis et al. [30, 70]. They argued that stochastic methods conceptually provide an easy way to test heuristics for page layout. However, it is probably fair to say that neither of these systems give rise to high-quality layouts.

Finally, we note the use of relational grammars to generate page layouts. They use a rule-based approach to generate essentially arbitrary placement constraints between the elements of the document. Weitzman and Wittenburg [79] pioneered this approach, which has since been extended by Kamps [44], Bateman et al. [6], Kong et al. [51] and di Iorio et al. [21].

4.4 Retargeting a page layout

Given the rise in popularity of mobile devices, the problem of automatically retargeting existing page layouts to small screen displays has become extremely relevant in recent years. Although a variety of solutions have been proposed, most of them adopt one of two main strategies. One approach is to create a scaled down version of the original page that retains the overall layout as much as possible while

[7]Available from `https://timesreader.nytimes.com/`

augmenting or adapting specific page elements to improve readability and usability. This is essentially a fine-tuning approach to the retargeting problem. The other strategy is to completely reformat the web page to produce a layout that is better suited to small screen viewing.

The problem of reformatting a page is conceptually similar to choosing a new page layout. However, rather than applying some of the more general layout strategies described in the previous subsection, most reformatting approaches impose a pre-defined page layout model that works well for small devices (e.g., the simple card metaphor used in the WEST system [11]) and then determine how to organize the original page content into that layout. This still basically boils down to a discrete problem in which the goal is to assign the relevant portions of the original content to one of the regions in the new small screen layout. A common approach to this assignment problem is to first analyze the original page to infer its semantic structure (e.g., what content is most important, what parts of the page are semantically related) and then use this structure to decide which chunks of the original content to include and where to position those chunks in the new layout. Recent techniques tend to use some combination of visual cues and semantic cues derived from the Document Object Model (DOM) of the original page to determine semantic structure. For example, Chen et al. [18] use heuristics based on common shapes and positions of semantically meaningful content blocks (e.g., headers, footers, navigation sidebars) to classify various parts of a page, while Björk et al. [11] use the presence of key page elements like headings, paragraphs, etc. to infer which pieces of content are related. In more recent systems [5, 80], low-level appearance characteristics such as colours and fonts are also taken into account to identify semantically coherent blocks of content.

In contrast to reformatting approaches, the main motivation behind rescaling techniques is to make the retargeted page look as much as possible like the original page. This strategy can make it easier for viewers to recognize familiar pages and leverages their spatial memory for accessing various parts of the page. The key problem with scaling down web pages is that important elements (e.g., text, images) can quickly become unintelligible. Some approaches to improving legibility adopt focus plus context techniques. For instance, O'Hara et al. [68] propose combining a uniformly scaled-down thumbnail view with a separate zoomed in detail view of the page in a single layout, while the Fishnet browser [7] enlarges a selected region of the page "in place" using a fisheye distortion technique. More recent work such as Minimap [72] and Summary Thumbnails [52] propose distortion-free methods for enlarging text with respect to the rest of the page to improve readability. The Summary Thumbnails approach also tries to reduce textual content (by removing less important words) when necessary to preserve the original line count of enlarged text.

5. TABLE FORMATTING

Tables are provided in virtually all document formatting systems. While tables could be considered as just another page layout model we have chosen to treat table formatting as a separate issue because of the many specialized algorithms that have been investigated. Like page layout it is natural to divide table formatting into two steps. The first step is to determine the *logical table structure* from the multi-dimensional data in the table. The logical structure specifies the number of columns and rows, the logical cells in the table and their content and the rows and columns they occupy. Determining the logical structure requires choosing which data dimension(s) are placed horizontally and which are placed vertically, how to nest these if more than one data dimension is placed on the same spatial dimension, and, in the case of categorical information, how to order the categories. To the best of our knowledge there has been no research into automatically determining the logical table structure from the underlying multi-dimensional data.[8] However, a variety of authors, including Vanoirbeek, and Wang and Wood [76, 77], have investigated editing models for changing the presentation structure while preserving the logical structure and

The next step is *table layout*. This finds a feasible layout for the table, i.e., a width for each column and height for each row such that each cell in the table is large enough to contain its content and the designer constraints – such as two columns have the same width – are satisfied. Usually the problem is to find a feasible layout satisfying an objective function such as minimizing the table height for some maximum allowed width.

Automatic layout of tables which contain text is computationally expensive. This is because if a cell contains text then this implicitly constrains the cell to take one of a discrete number of possible *configurations* corresponding to different numbers of lines of text. These lead to an exponential number of different possible configurations for the table. For this reason most document formatting systems require the author to give the width for a column or else compute the width of a column by laying out the content of each cell on a single line and setting the column width to the widest of these. Given the column widths, the row heights are computed in the obvious way: the height of each cell is computed by taking the maximum height of a row in a column. This is straightforward to extend to compound rectangular cells. However, for on-line presentation or variable content it is not practical to require the author to specify the column widths at document authoring time. Thus there has been renewed interest in automatic table formatting.

5.1 Column-driven layout

A simple approach is to compute an ideal width for each unknown width column (such as the maximum of the width of each cell in the column when its content is laid out on a single line) and a minimum width which is the maximum of the minimum width for each cell in the column. Then, using a variant of the TEX algorithm for line justification, compute an adjustment ratio for the table which proportionately scales down the unfixed columns from their ideal size (but not below their minimum size) until the table is narrow enough. Now given the column widths, compute the row heights by laying out content of the cells in each row. The effect is to minimize the table height for a particular width, but, since text cell configurations are not computed and the two dimensional interaction between columns is ignored, the method is a fairly coarse approximation.

The suggested automatic layout algorithm for HTML tables [71] works like this. It also allows the designer to specify that a column should have width that is some fixed ratio of

[8]Although we note that HTML was designed to allow some automated transformations for braille and for speech [14].

the special width "$*$". The related approach of [13, 3] allows the designer to specify required and preferred linear arithmetic constraints over column widths. A linear constraint solver is then used to determine column widths. Lutteroth and Weber [60] also allow linear constraints over column widths in their extension of standard table layout which allows columns to be partially ordered rather than totally ordered.

5.2 Cell-driven layout

Better table layout requires modelling the two-dimensional interaction between cells and the different configurations for text cells. Beach [8] was the first to consider automatic table layout from a constrained optimization viewpoint. He considered packing rectangular fixed sized cells into a grid and showed this can be done in polynomial time. He also consider packing unordered elements into a minimum space and showed this was NP-complete. He did not consider the case where cells could contain text with multiple line breaking choices. Wang and Wood [77] presented a branch and bound algorithm, accelerated with a polynomial-time greedy algorithm for finding a table layout satisfying linear designer constraints on the column widths and row heights with multiple line break choices in each cell. They showed that the associated decision problem was NP-Hard. They modelled table layout as a satisfaction problem rather than an optimization problem.

Anderson and Sobti [1] showed that finding the minimum height layout for a fixed maximum width is NP-Complete for simple tables even without designer constraints. They also proved that minimising a non-zero linear combination of overall width and height is computable in polynomial time. They gave two heuristic methods for finding a minimum height layout for simple cells. The first was based on encoding table layout as the problem of finding the minimum cut in a flow graph while the second is a continuous approximation to the problem in which the convex hull of the configurations is modelled using a conjunction of linear inequality constraints. They found that the approach based on finding a minimum cut in a flow graph performed better than the convex hull approach. However, encoding table layout as a flow graph is complex and it is not clear how this approach can be generalised to handle tables with compound cells or designer constraints.

Using a continuous linear approximation to the constraint that a cell is large enough to contain its content has been suggested by a number of other researchers such as Lin [55]. The advantage of the linear approximation that it naturally generalises to handle compound cells and linear designer constraints. The main disadvantage is that it can lead to a large linear program that require the use of sophisticated linear programming software.

Beaumont [9] and Hurst et al. [38] suggested a non-linear continuous approximation in which the area of each cell is constrained to be greater than the area of its content (when laid out in a single line). Beaumont used the non-linear solver MINOS to solve the resulting non-linear problem while Hurst et al. noted that it was a convex optimization problem and could be modelled using conic programming and solved using polynomial time interior point methods. However, the use of sophisticated and slow non-linear optimization software is not that practical. Hurst et al. [36] have given a more efficient specialized variable elimination method for solving a simplified form of the continuous approximation. Unfortunately, this does not handle compound cells or lower bounds on cell height and width.

Hurst et al. [38] also gave a simple hill-climbing local search method for table layout. The algorithm starts from the narrowest layout for the table and iteratively reduces the height of a row (and hence the table) by choosing to narrow the row for which this will lead to the least increase in width (for a given height reduction).

5.3 Minimal configurations

The main decision in table layout is how to break the lines of text in each cell. Different line breaks give rise to different cell configurations. Many of the above table layout methods require computation of minimal width/height configurations. The difficult issue is to compute the minimum width for a fixed height rectangle large enough to contain its textual content. This is an example of a one-dimensional minimization problem, assuming that the cell content satisfies the *anti-monotonicity* property that the height required to fit the cell content decreases as the width of the cell increases.

The problem of determining the minimum width for a fixed height rectangle was first considered in Anderson and Sobti [1]. A simple, generic approach is to use binary or secant based search to find the minimum width and to determine the minimum height for a particular width w simply lay out the text in a box of that width greedily placing text on the line until it is full. This gives rise to a $O(W \log N)$ time algorithm where W is the number of words and N the length of the total text. This approach was suggested in Anderson and Sobti and some improvements are described in [39].

Other more efficient algorithms for finding the minimum width have been developed. However they are quite complex and assume that the box contains uniform height text with no complex formatting. Thus Hurst et al. [38] gave a $O(W)$ dynamic programming algorithm where W is the number of words while Anderson and Sobti gave an $O(N^{1/2} \log N)$ time algorithm where N is the length of the total text. However the Anderson and Sobti algorithm requires a significant once-off pre-computation that takes, as best we understand, $O(N^{3/2} \log N)$ time.

Hurst et al. [39] generalized the problem of finding a minimal width or height rectangle that is large enough to contain its content to more complex shapes. They modelled the shape using a trapezoid list where the vertices of the trapezoids are a linear function of a single variable α and the area of the shape increases monotonically in α. Since this is a one-dimensional minimization problem they used secant search. This is in some sense the dual problem to scaling the textual content to fit a fixed size shape which was discussed in Section 4.2.

6. CONCLUSIONS

Our review of automatic document formatting has hopefully demonstrated that it is useful to view it has a number of different but linked constrained optimization problems arising in micro-typography, macro-typography and table layout. We have seen that the maturity of techniques to solve these problems varies greatly. On one hand solutions to micro-typographical issues such as line breaking and justification are efficient and robust enough to be used in practical

applications. On the other hand sophisticated optimization techniques for the macro-typographical issue of finding a page layout are not used in practical applications—the automatic generation of yellow pages directories being a notable exception. The practicality of current techniques for automatic table formatting lies somewhere in between.

We believe that the industrial importance of Variable Data Printing and the increasing sophistication of on-line formatting – for example, multi-column and grid layout – will drive the development of better automatic techniques for page layout in the next decade. Almost certainly these will utilize incomplete constructive search or local search techniques. One issue that will need to be addressed is how to combine automatic micro- and macro-typographic layout technique. The difficulty is that the layout issues interact, but solving the entire problem seems prohibitively expensive.

Other issues that will need to be addressed are better support for authoring documents that can take advantage of better automatic layout. Also better understanding and support for the kinds of interaction that electronic documents offer.

7. ACKNOWLEDGEMENTS

We would like to thank Simon Towers, Grayson Lang and Bert Bos for their suggestions and criticisms. Marriott acknowledges the support of ARC through the Discovery Project Grant DP0987168

8. REFERENCES

[1] R. J. Anderson and S. Sobti. The table layout problem. In *COMPGEOM: Annual ACM Symposium on Computational Geometry*, pages 115–123, 1999.

[2] G. Asher. Inside type & set. *TUGBoat*, 13, 1992.

[3] G. J. Badros, A. Borning, K. Marriott, and P. Stuckey. Constraint cascading style sheets for the web. In *Proceedings of the 1999 ACM Conference on User Interface Software and Technology*, pages 73–82, New York, Nov. 1999. ACM.

[4] G. J. Badros, J. J. Tirtowidjojo, K. Marriott, B. Meyer, W. Portnoy, and A. Borning. A constraint extension to scalable vector graphics. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 489–498, New York, NY, USA, 2001. ACM Press.

[5] S. Baluja. Browsing on small screens: recasting web-page segmentation into an efficient machine learning framework. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 33–42, New York, NY, USA, 2006. ACM.

[6] J. Bateman, J. Kleinz, T. Kamps, and K. Reichenberger. Towards constructive text, diagram, and layout generation for information presentation. *Comput. Linguist.*, 27(3):409–449, 2001.

[7] P. Baudisch, B. Lee, and L. Hanna. Fishnet, a fisheye web browser with search term popouts: a comparative evaluation with overview and linear view. In *AVI '04: Proceedings of the working conference on Advanced visual interfaces*, pages 133–140, New York, NY, USA, 2004. ACM.

[8] R. J. Beach. *Setting tables and illustrations with style*. PhD thesis, University of Waterloo, 1985.

[9] N. Beaumont. Fitting a table to a page using non-linear optimization. *Asia-Pacific Journal of Operational Research*, 21(2):259–270, 2004.

[10] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, September 1999.

[11] S. Björk, L. E. Holmquist, J. Redström, I. Bretan, R. Danielsson, J. Karlgren, and K. Franzén. West: a web browser for small terminals. In *UIST '99: Proceedings of the 12th annual ACM symposium on User interface software and technology*, pages 187–196, New York, NY, USA, 1999. ACM.

[12] A. Borning, R. Lin, and K. Marriott. Constraints for the web. In *Proceedings of ACM MULTIMEDIA'97*, pages 173–182, Nov. 1997.

[13] A. Borning, R. Lin, and K. Marriott. Constraint-based document layout for the web. *Multimedia Systems*, 8(3):177–189, 2000.

[14] B. Bos. Personal communication, May 2009.

[15] B. Bos, H. Lie, C. Lilley, and I. Jacobs. Cascading Style Sheets, level 2 CSS2 Specification. W3C Recommendation. http://www.w3.org/TR/REC-CSS2, 1998.

[16] C. Braganza, K. Marriott, P. Moulder, M. Wybrow, and T. Dwyer. Scrolling behaviour with single- and multi-column layout. In *ACM Conference on the World Wide Web (WWW 2002)*, pages 831–840, 2009.

[17] A. Brüggemann-Klein, R. Klein, and S. Wohlfeil. Pagination reconsidered. In *Electronic Publishing*, volume 8, pages 139–152, 1995.

[18] Y. Chen, W.-Y. Ma, and H.-J. Zhang. Detecting web page structure for adaptive viewing on small form factor devices. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 225–233, New York, NY, USA, 2003. ACM.

[19] J. B. S. de Oliveira. Two algorithms for automatic document page layout. In *DocEng '08: Proceeding of the eighth ACM symposium on Document engineering*, pages 141–149, New York, NY, USA, 2008. ACM.

[20] J. D. DeTreville. *An Analytical Approach to Computerized News Layout for Newspapers*. PhD thesis, MIT, Cambridge, MA, USA, 1978.

[21] A. Di Iorio, L. Furini, F. Vitali, J. Lumley, and T. Wiley. Higher-level layout through topological abstraction. In *DocEng '08: Proceeding of the eighth ACM symposium on Document engineering*, pages 90–99, New York, NY, USA, 2008. ACM.

[22] D. Eppstein, Z. Galil, R. Giancarlo, and G. F. Italiano. Sparse dynamic programming ii: convex and concave cost functions. *J. ACM*, 39(3):546–567, 1992.

[23] S. K. Feiner. A grid-based approach to automating display layout. In *Proceedings on Graphics interface '88*, pages 192–197, Toronto, Ont., Canada, Canada, 1988. Canadian Information Processing Society.

[24] R. Furuta. Important papers in the history of document preparation systems: basic sources. *Electronic Publishing*, 5:19–44, 1992.

[25] R. Furuta, J. Scofield, and A. Shaw. Document formatting systems: survey, concepts, and issues. *ACM Computing Surveys (CSUR)*, 14(3):417–472, 1982.

[26] F. Glover and G. Kochenberger. *Handbook of metaheuristics*. Springer, 2003.

[27] E. Goldenberg. Automatic layout of variable-content print data. Technical Report 286, Hewlett-Packard Laboratories, Oct. 2002.

[28] W. H. Graf. The constraint-based layout framework laylab and its applications. In *In Proceedings of ACM Workshop on Effective Abstractions in Multimedia, Layout and Interaction*. ACM, 1995.

[29] W. H. Graf, S. Neurohr, and R. Goebel. Ypps—a constraint-based tool for the pagination of yellow-page directories. In *Proceedings of the KI-96 Workshop on Declarative Constraint Programming*, pages 87–97, 1996.

[30] S. J. Harrington, J. F. Naveda, R. P. Jones, P. Roetling, and N. Thakkar. Aesthetic measures for automated document layout. In *DocEng '04: Proceedings of the 2004 ACM symposium on Document engineering*, pages 109–111, New York, NY, USA, 2004. ACM.

[31] W. Hegazy and J. Gourlay. Optimal line breaking in music. In *Document Manipulation and Typography: Proceedings of the International Conference on Electronic Publishing, Document Manipulation, and Typography*, pages 157–169. Cambridge University Press, 1988.

[32] R. D. Hersch and C. Bétrisey. Method for producing visually evenly spaced typographic characters. US Patent, 1996.

[33] D. S. Hirschberg and L. L. Larmore. The least weight subsequence problem. *SIAM J. Comput.*, 16(4):628–638, 1987.

[34] N. Hurst. *Better automatic layout of documents*. PhD thesis, Monash University, Department of Computer Science, May 2009.

[35] N. Hurst and K. Marriott. Satisficing scrolls: a shortcut to satisfactory layout. In *DocEng '08: Proceeding of the eighth ACM symposium on Document engineering*, pages 131–140, New York, NY, USA, 2008. ACM.

[36] N. Hurst, K. Marriott, and D. Albrecht. Solving the simple continuous table layout problem. In *DocEng '06: Proceedings of the 2006 ACM symposium on Document engineering*, pages 28–30, New York, NY, USA, 2006. ACM Press.

[37] N. Hurst, K. Marriott, and P. Moulder. Cobweb: A constraint-based web browser. In M. J. Oudshoorn, editor, *ACSC*, volume 16 of *CRPIT*, pages 247–254. Australian Computer Society, 2003.

[38] N. Hurst, K. Marriott, and P. Moulder. Toward tighter tables. In *DocEng '05: Proceedings of the 2005 ACM symposium on Document engineering*, pages 74–83, New York, NY, USA, 2005. ACM Press.

[39] N. Hurst, K. Marriott, and P. Moulder. Minimum sized text containment shapes. In *DocEng '06: Proceedings of the 2006 ACM symposium on Document engineering*, pages 3–12, New York, NY, USA, 2006. ACM Press.

[40] C. Jacobs, W. Li, and D. Salesin. Adaptive document layout via manifold content. In *Proceedings of Workshop on Web Document Analysis*, Edinburgh, 2003.

[41] C. Jacobs, W. Li, E. Schrier, D. Bargeron, and D. Salesin. Adaptive grid-based document layout. *ACM Trans. Graph.*, 22(3):838–847, 2003.

[42] R. Johari, J. Marks, A. Partovi, and S. Shieber. Automatic yellow-pages pagination and layout. *Journal of Heuristics*, 2:321–342, 1997.

[43] M. Jourdan, N. Layaïda, C. Roisin, L. Sabry-Ismaïl, and L. Tardif. Madeus, and authoring environment for interactive multimedia documents. In *Proceedings of the sixth ACM international conference on Multimedia*, pages 267–272. ACM New York, NY, USA, 1998.

[44] T. Kamps and K. Reichenberger. Automatic layout based on formal semantics. In *AVI '94: Proceedings of the workshop on Advanced visual interfaces*, pages 231–233, New York, NY, USA, 1994. ACM.

[45] H.-K. Kan. *A Computerized Template-driven News-layout System for Newspapers*. PhD thesis, MIT, Department of Electrical Engineering, 1977.

[46] P. Karow. Two Decades of Typographic Research at URW: A Retrospective. In *Electronic Publishing, Artistic Imaging, and Digital Typography '98*, volume 1190 of *LNCS*, pages 265–280. Springer-Verlag, 1998.

[47] B. W. Kernighan. A troff tutorial, unix version 7 manual. Technical report, Bell Laboratories Computing Science, June 1978.

[48] D. E. Knuth. *The TEXbook*. Addison-Wesley, Reading, Massachusetts, 1984.

[49] D. E. Knuth. *Digital Typography*. Cambridge University Press, New York, NY, USA, 1997.

[50] D. E. Knuth and M. F. Plass. Breaking paragraphs into lines. In *Software—Practice and Experience, 11(11)*, pages 1119–1184, Nov. 1982.

[51] J. Kong, M. Qiu, and K. Zhang. Authoring multimedia documents through grammatical specifications. In *ICME '03: Proceedings of the 2003 International Conference on Multimedia and Expo*, pages 629–632, Washington, DC, USA, 2003. IEEE Computer Society.

[52] H. Lam and P. Baudisch. Summary thumbnails: readable overviews for small screen web browsers. In *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 681–690, New York, NY, USA, 2005. ACM.

[53] J. Lau and P. Karow. hz-program: Micro-typography for advanced typesetting. In *SID Symposium 1993*, pages 58–62, 1993.

[54] H. W. Lie. The electronic broadsheet: all the news that fits the display, 1991. SM Thesis, MIT Media Lab.

[55] X. Lin. Active layout engine: Algorithms and applications in variable data printing. *Computer-Aided Design*, 38(5):444–456, 2006.

[56] X. Lin. Predictive Text Fitting. *Lecture Notes in Computer Science*, 4073:13–23, 2006.

[57] G. Loureiro and F. Azevedo. Constrained xsl formatting objects for adaptive documents. In *DocEng '05: Proceedings of the 2005 ACM symposium on Document engineering*, pages 95–97, New York, NY, USA, 2005. ACM Press.

[58] J. Lumley, R. Gimson, and O. Rees. Extensible layout in functional documents. In *Proceedings of SPIE*, volume 6076, pages 177–188, 2006.

[59] J. Lumley, R. Gimson, and O. Rees. Resolving layout interdependency with presentational variables. In *DocEng '06: Proceedings of the 2006 ACM symposium on Document engineering*, pages 95–97, New York, NY, USA, 2006. ACM.

[60] C. Lutteroth and G. Weber. User interface layout with ordinal and linear constraints. In *AUIC '06: Proceedings of the 7th Australasian User Interface Conference*, pages 53–60. Australian Computer Society, Inc., 2006.

[61] K. Marriott, B. Meyer, and L. Tardif. Fast and efficient client-side adaptivity for svg. In *ACM Conference on the World Wide Web (WWW 2002)*, 2002.

[62] K. Marriott, P. Moulder, and N. Hurst. Automatic float placement in multi-column documents. In *DocEng '07: Proceedings of the 2007 ACM symposium on Document engineering*, New York, NY, USA, 2007. ACM Press.

[63] C. McCormack, K. Marriott, and B. Meyer. Adaptive layout using one-way constraints in SVG. In *Proceedings of third Annual Conference on Scalable Vector Graphics, SVG Open*, 2004.

[64] F. Mittelbach. Formatting documents with floats – a new algorithm for LaTeX 2_ε*. *TUGboat*, 21, 2000.

[65] F. Mittelbach and C. Rowley. The pursuit of quality—how can automated typesetting achieve the highest standards of craft typography? In *EP92 (Proceedings of Electronic Publishing)*, pages 261–273. Cambridge University Press, 1992.

[66] H.-W. Nienhuys and J. Nieuwenhuizen. Lilypond, a system for automated music engraving. In *Proceedings of the XIV Colloquium on Musical Informatics (XIV CIM 2003)*, Firenze, Italy, May 2003.

[67] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, 1999.

[68] K. O'Hara, A. Sellen, and R. Bentley. Supporting memory for spatial location while reading from small displays. In *CHI '99: CHI '99 extended abstracts on Human factors in computing systems*, pages 220–221, New York, NY, USA, 1999. ACM.

[69] M. F. Plass. *Optimal pagination techniques for automatic typesetting systems*. PhD thesis, Stanford University, June 1981.

[70] L. Purvis, S. Harrington, B. O'Sullivan, and E. C. Freuder. Creating personalized documents: an optimization approach. In *DocEng '03: Proceedings of the 2003 ACM symposium on Document engineering*, pages 68–77, New York, NY, USA, 2003. ACM Press.

[71] D. Raggett, A. L. Hors, and I. Jacobs. HTML 4.01 Specification, section 'Autolayout Algorithm'. http://www.w3.org/TR/html4/appendix/notes.html#h-B.5.2, 1999.

[72] V. Roto, A. Popescu, A. Koivisto, and E. Vartiainen. Minimap: a web page visualization method for mobile phones. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 35–44, New York, NY, USA, 2006. ACM.

[73] S. Russell and P. Norvig. *Artificial Intelligence: a Modern Approach*. Prentice Hall, 2nd edition, 2002.

[74] E. Schrier, M. Dontcheva, C. Jacobs, G. Wade, and D. Salesin. Adaptive layout for dynamically aggregated documents. In *Proc. of the 13th Intl. Conf. on Intelligent User Interfaces*, pages 99–108. ACM, 2008.

[75] H. T. Thành. Micro-typographic extensions to the TeX typesetting system (doctoral dissertation). *TUGBoat*, 21, 2000.

[76] C. Vanoirbeek. Formatting structured tables. In *EP92 (Proceedings of Electronic Publishing)*, pages 291–309. Cambridge University Press, 1992.

[77] X. Wang and D. Wood. Tabular formatting problems. In *3rd Principles of Document Processing*, pages 171–181, 1996.

[78] L. Weitzman and K. Wittenburg. Relational grammars for interactive design. In *IEEE Symposium on Visual Languages*, pages 4–11, 1993.

[79] L. Weitzman and K. Wittenburg. Automatic generation of multimedia documents using relational grammars. In *Proceedings of 2nd ACM Conference on Multimedia*, 1994.

[80] X. Xie, G. Miao, R. Song, J.-R. Wen, and W.-Y. Ma. Efficient browsing of web search results on mobile devices based on block importance model. In *PERCOM '05: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications*, pages 17–26, Washington, DC, USA, 2005. IEEE Computer Society.

[81] H. Zapf. About micro-typography and the *hz*-program. *Electronic Publishing*, 6(3):283–288, Sept. 1993.