

A Probabilistic Line Breaking Algorithm

Remco R. Bouckaert

Xtal Mountain Information Technology and Computer Science Department,
University of Waikato, New Zealand
rrb@xm.co.nz, remco@cs.waikato.ac.nz

Abstract. We show how a probabilistic interpretation of an ill defined problem, the problem of finding line breaks in a paragraph, can lead to an efficient new algorithm that performs well. The graphical model that results from the probabilistic interpretation has the advantage that it is easy to tune due to the probabilistic approach. Furthermore, the algorithm optimizes the probability a break up is acceptable over the whole paragraph, it does not show threshold effects and it allows for easy incorporation of subtle typographical rules. Thanks to the architecture of the Bayesian network, the algorithm is linear in the number of characters in a paragraph. Empirical evidence suggests that this algorithm performs closer to results published through desk top publishing than a number of existing systems.

1 Introduction

With the advance of information systems, the amount of automatically formatted text increases daily. However, judging from a software patent search, not a lot of attention is spent on increasing the quality of layout. New interest in this area is driven by the standardization of style specifications through CSS and XSL-FO [8]. Limitations of automatic typesetting systems are partly due to the fact that the underlying algorithms have been designed with resource restrictions in mind. For example, once \TeX [3] has sufficient material to complete a page, the page is output to free up memory. If many floating objects like figures and tables are present, these can be placed in undesirable locations far from the place where they are referenced.

Considerations of limited resources were quite valid in the time that these algorithms were developed. However, they do not apply today and this opens a way to improve various typesetting algorithms. In typesetting a wide range of decisions need to be made in the areas of hyphenation, line breaking, page breaking, floating object placement, footnotes, headers and footers, page numbering, reference resolution, etc. In this paper, we consider line breaking only, but the principles presented should be extendible to page breaking, floating object placement and other areas as well.

A reasonable approach seems to be to ask professional publishers what makes a good line break-up. However, this knowledge is very difficult to elicit. Typically only examples of what is acceptable and what is not can be given, with

disagreement on borderline cases. So finding good line break-ups is an ill specified problem in which decisions need to be made with uncertainty of the function to optimize.

In this paper we take a probabilistic approach to line breaking in which we model the probability that a particular break-up is acceptable. The probability of acceptability can be maximized and the break-up with the highest probability chosen. A dynamic graphical model is used to capture acceptability.

The next section introduces concepts of line breaking and related work. In Section 3 a probabilistic algorithm is presented for breaking lines. Section 4 compares the performance of some popular typesetting systems with data published through desk top publishing together with an empirical comparison of the new algorithm. We finish with concluding remarks and suggestions for further extensions in Section 5.

2 The line breaking problem

Breaking a paragraph into lines consists of selecting the break points in a paragraph. To determine how much space a line takes, each character (or rather glyph, see comment on ligatures below) in the paragraph is modeled as a box with a specific width, height and elevation from the baseline. These properties are determined by the font being used and the font size. The length of a line is the sum of the widths of the boxes in the line. The width of a box is not only determined by the width of the character, but is also influenced by the following character. If the width of a character would not be *kerned*, the result may look rather unpleasing. For example, compare these unknerved and knerved words:

BAYES
BAYES

The latter has less space between the A and Y, and the width of the A is modeled less wide then when an A is followed by, say, another A.

Some fonts treat certain combinations of characters as a single character. Examples of such so called *ligatures* is the combination of f and i and the combination of two f's and an i. The following line illustrates the difference between separate characters and their ligatures:

efficient fix
efficient fix

So, the first step in breaking a line is determining the width of its characters, taking kerning and ligatures in account. The next step is determining possible break points. Typical break points in Latin alphabet languages are whitespaces, hyphen characters and points where hyphenation is allowed.

Finding these hyphenation points by maintaining a wordlist is cumbersome because of the many variations of appearances of a word (e.g. singular and plural). Furthermore, maintaining such a word list in areas where new words are

invented regularly (e.g. brand names) is a huge task. An elegant alternative is to use a hyphenation algorithm [3, 6] which essentially works as follows. Some words are stored in an exception list (for example, as-so-ciate). For words not in the exception list, a pattern list is used, which is generated from a word list containing hyphenated words. Patterns contain characters and numbers, for example 'y3thin'. A word is compared with the characters in the patterns and every character is assigned the highest number in the pattern. The word 'anything' matches 'ythin' in the pattern 'y3thin' and the y gets assigned the number 3. Any odd numbered character is a hyphenation point. So 'any-thing' is allowed. It is claimed [3] that more than 95% of the allowable hyphenation points in US English can be found by this algorithm (taking word frequencies in account).

Hyphenation points at the first λ and last ρ characters are discarded to prevent unacceptable hyphenation points. For example, consider the hyphenation points found when not suppressing the first and last points in the words 'a-ha', 'Bal-i', 's-tu-den-t' or 'A-pu-li-a'. By default, λ is 2 and ρ is 3 for English.

A broken line may not completely fit into the available space between the left and right margin. To make it fit, space may be distributed among the whitespaces in the line, or some whitespace may be taken out if the text exceeds the text width. We assume that there is a single target width for the complete paragraph and the paragraph is adjusted. An indent that applies to the first line of the paragraph can be modeled using a fixed width unbreakable space (possibly with a negative value).

Now that we have a flavor of the problem, we can specify some terms more formally. A *paragraph* p is a sequence of $n > 0$ characters c_i , $1 \leq i \leq n$. A *break point candidate* in p is an index of a character in p for which it is allowed to break a line. Typical break point candidates are space and hyphen characters and hyphenation points in words. We are interested in finding a sequence of character indexes s_0, s_1, \dots, s_k , $\forall 1 \leq j \leq k$ $1 \leq s_j \leq n$ and $s_{j-1} < s_j$ where each s_j is a break point candidate. Each of the indices s_j indicates a *break point* in p . Index s_0 is added for convenience of further definitions and by convention $s_0 = 0$ and $s_k = n$. A *line* in p with break points s_0, \dots, s_k is the sub-sequence of characters between breakpoints, more specific for $1 \leq j \leq k$, $l_j = \{c_{s_{j-1}+1} \dots c_{s_j}\}$ (note $s_0 = 0$ so $l_1 = \{c_1 \dots c_{s_1}\}$). So, a break point s_j is the last character on line l_j .

The *character width* cw_i $1 \leq i \leq n$ of a character c_i in a paragraph p with break points s_0, \dots, s_k is the width of the character taking the font properties in account. Furthermore, the character width takes breakpoints in account with the following rules:

- A character not on a break point may have its width corrected due to kerning.
- A space character at the beginning or end of a line has width 0.
- A non-space and non-hyphen character not at the end of a word that is a break point has width of the character plus the width of a hyphen.
- If c_i is part of a ligature, and the ligature is not broken, cw_i is the width of the ligature divided among the characters that make up the ligature.

The *natural line width* nw_i , $0 \leq i \leq n$ at character c_i is the sum of widths of characters on to the closest previous breakpoint s_j . More specific, nw_0 is the

indent, and if $s_j = 0$, $nlw_i = nlw_0 + \sum_{d=1}^i cw_d$, otherwise $nlw_i = \sum_{d=s_j+1}^i cw_d$. The *natural line width* of a line l_j , $1 \leq j \leq k$ is nlw_{s_j} . The *number of spaces* in a line l_j is $nsp_{s_j} = \sum_{i=s_{j-1}+1}^{s_j-1} \{1|c_i = \text{space}\}$, so spaces at the start and end of a line are not counted (it is assumed no space follows another space).

The *line breaking problem* for a paragraph p for a desired *text width* and *indent* is finding a set of break points of p that look 'nice'. We will consider fixed text width, tw , throughout this paper with the exception of the first line, which can have a (possibly negative) indent in .

A simple approach to the line breaking problem is to consider lines one by one, like Unix's Troff. Simply keep adding words as long as the natural line width does not exceed the text width. If a word makes the line overflow, try hyphenate the word (using a word list or Liang's hyphenation algorithm) and add as many parts (possibly zero) of the word to the line as long as the text width is not exceeded. Output the line and place the remainder on the following line. In this approach, line breaks are only optimized locally, without regard to the breaks in the remainder of the paragraph.

\TeX 's line breaking algorithm [4] on the other hand optimizes line breaks on the level of the paragraph. We give a short description of \TeX 's line breaking algorithm here (see [3] for an extensive description). \TeX determines character widths taking kerning and ligatures in account and uses a three phased process.

In phase 1, no hyphenation points are considered for line breaking, only white spaces. For a paragraph broken in k lines, for each line $j = 1 \dots k$ a badness b_j is calculated, using $b_j = 100 \left(\frac{nlw_{s_j} - tw}{nsp_{s_j} \cdot f} \right)^3$ where nlw_{s_j} the natural line width, tw the text width, nsp_{s_j} the number of spaces of line j and f is a factor that differs depending on whether the line needs to be stretched or shrunk. Note that $nlw_{s_j} - tw$ represents the amount that the line needs to be stretched or shrunk. Depending on the value of b_j , the line is classified as tight, decent, loose or very loose. If none of the line's badnesses exceed a 'pretolerance' limit, the paragraph is accepted.

Otherwise, in phase 2, the hyphenation candidates are calculated for the paragraph and a new set of breakpoints is calculated. If all of the line's badnesses are below a tolerance level (which differs from the pretolerance level) a demerit for the paragraph is calculated as a combination of line badnesses, roughly as

$$\sum_{j=1}^n (c + b_j)^2 + p \cdot |p| \tag{1}$$

where c is a constant line penalty and p a penalty term. A tight line next to a loose line increases the penalty p . If two consecutive lines are hyphenated, or if the last broken line is hyphenated, the penalty is increased. A paragraph with a demerit below a threshold is accepted.

Otherwise, in phase 3, the steps in phase 2 are repeated but now with an emergency stretch that allows lines to shrink or expand more. If this last phase does not succeed, \TeX outputs overly long lines, due to the thresholds in the algorithm.

3 A probabilistic line breaking algorithm

We start with deriving the line breaking algorithm and illustrate how to design a simple but efficient probabilistic algorithm based on a dynamic Bayesian network. Then, this model is generalized and we show how the behavior of the network can be influenced by tuning network parameters. Because the network is easy to comprehend, the resulting change in behavior is intuitively satisfying.

3.1 A simple probabilistic line breaking algorithm

We approach the line breaking problem by defining a probability distribution that represents the acceptability of a set of breakpoints.

Let p be a paragraph with n characters c_1, \dots, c_n , text width tw , indent in and a set of break points $S = \{s_0, \dots, s_k\}$. Let a_i , $1 \leq i \leq n$ be a boolean representing whether the break-up up to character i is acceptable, and let a denote a_n . Then we are interested in $P(a, S, p, in)$ where we leave tw and in implicit. We make the following two assumptions

1. the acceptability of the first i characters in the break-up is independent of the break-up of the remainder, provided nlw_{i-1} is known for the remainder.
2. we have no initial preference for any breakup, that is, $P(S|p, in)$ is constant for any S .

The first assumption is the base for the efficiency of \TeX 's line breaking algorithm. The last assumption is for the sake of simplifying the derivation of the probabilistic algorithm, and will be relaxed in the following section. Now, $P(a, S, p, in) = P(a|S, p, in)P(S|p, in)P(p, in)$. Since p and in are given, and using the second assumption, $P(p, in)$ and $P(S|p, in)$ are constant, so $P(a, S, p, in) \propto P(a|S, p, in)$. The first assumption lets us decompose $P(a|S, p, in)$ as $P(a_i|S \setminus \{i+1, \dots, n\}, p, in) \cdot P(a|S \setminus \{1, \dots, i\}, p, nlw_i, in)$, where the first term is the acceptability of the break-up of the first i characters, and the second term the acceptability of the remainder given the natural line width nlw_i of the last line of the first i characters. Now note $P(a_i|S \setminus \{i+1, \dots, n\}, p, in) = P(a_i|S \setminus \{i+1, \dots, n\}, p \setminus \{c_{i+1}, \dots, c_n\}, in)$. Therefore, we can write $P(a|S, p, in)$ as the product of the acceptability of the individual characters.

That is $P(a|S, p, in)$ equals $P(a_1|S \setminus \{2, \dots, n\}, p, in) \cdot P(a|S \setminus \{2, \dots, n\}, p \setminus \{c_2, \dots, c_n\}, nlw_1, in)$. Observe, in the first term $S \setminus \{2, \dots, n\}$ can be interpreted as the boolean $1 \in S$ and that in the latter term the indent does not give any extra information once nlw_1 is known. So, we can write $P(a_1|1 \in S, p, in) \cdot P(a|S \setminus \{2, \dots, n\}, p \setminus \{c_2, \dots, c_n\}, nlw_1)$. Repeating this process and realising $nlw_0 = in$ gives $P(a|S, p, in) = \prod_{i=1}^n P(a_i|i \in S, p, nlw_{i-1})$.

Further, the natural line width at character i , $0 \leq i \leq n$ is calculated as

$$nlw_i = \begin{cases} cw_i & | i - 1 \in S \\ nlw_{i-1} + cw_i & | i - 1 \notin S \\ in & | i = 0 \end{cases}$$

Now we are left with determining $P(a_i|i \in S, p, nlw_{i-1})$. If c_i is not a break point candidate (i.e., it is not a space, hyphen, or hyphenation point) and $i \in S$, the break-up is not acceptable under any circumstances, and $P(a_i|i \in S, p, nlw_{i-1}) = 0$. If not breaking at c_i means that the natural line width remains within the text width, the line break-up is just as acceptable as the acceptability of the remainder of the paragraph, $P(a_i|i \in S, p, nlw_{i-1}) = 1$. However, not breaking may be unacceptable as well, if this means extending the natural length beyond the text width. Breaking before the natural line width has reached the text width also decreases acceptability. In summary, $P(a_i|i \in S, p, nlw_{i-1})$ is defined as

$$\begin{array}{ll}
0 & |i \text{ is not a break point candidate and } i \in S \\
1 & |nlw_{i-1} + cw_i \leq tw \text{ and } i \notin S \\
P^a(cw_i|nlw_{i-1} - tw) & |nlw_{i-1} + cw_i > tw \\
P^a(cw_i|nlw_{i-1} - tw) & |nlw_{i-1} + cw_i \leq tw \text{ and } i \in S
\end{array}$$

where P^a a distribution representing acceptability for breaking not earlier than at c_i instead of c_{i-1} . For the choice of this distribution, we let ourselves inspire by the empirical distribution of what seems to be acceptable as shown in Figure 2 by the graph for published line lengths. This shows a distribution which is roughly normal, with different variances for lines exceeding and undercutting the text width. So, we define

$$P^a(cw_i|x) = \begin{cases} N(x + cw_i, \sigma^+) / P^a(x) & |x + cw_i \geq 0 \\ N(x + cw_i, \sigma^-) / P^a(x) & |x + cw_i < 0 \end{cases}$$

where $N(x, \sigma)$ the standard normal distribution $N(x, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$, $P^a(x) = N(x, \sigma^+)$ if $x \geq 0$ and $N(x, \sigma^-)$ otherwise and σ^+ and σ^- two variances, which are the two parameters to choose. Note that the contribution of line j to $P(a|S, p, in)$ is $\prod_{i=s_{j-1}+1}^{s_j} P(i \in S|p, nlw_{i-1})$ and that this reduces to $N(nlw_{s_j} - tw, \sigma^+)$ if line j 's natural width exceeds the text width ($nlw_{s_j} > tw$) and $N(nlw_{s_j} - tw, \sigma^-)$ if $nlw_{s_j} \leq tw$.

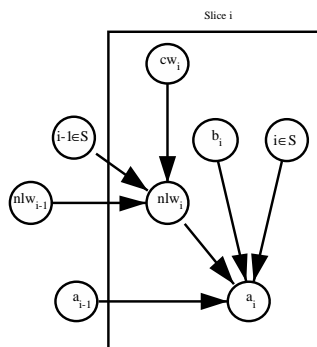
What we are after is of course selecting a set of break points that maximizes $P(a|S, p, in)$. Note that what we have done so far is derive a dynamic Bayes network with a structure as pictured in Figure 1. The character widths cw_i are given (refer to the rules as described in Section 2 for the actual details). Node b_i indicates whether breaking is allowed at character c_i . The conditional distribution for nlw_i and a_i are described in this section above.¹ What remains is selecting a value for the boolean variable indicated by $i \in S$ in the figure. Straightforward inference e.g. as described by Pearl [7] or Lauritzen [5], for finding the most likely configuration for b_i suffices to find the breakpoints.

It is interesting to see how this algorithm relates to T_EX's line breaking algorithm. It can be shown² for a paragraph p that a set of break points $S =$

¹ Instead of implementing one slice per character, one slice per break point candidate can be used. Of course, a correction for the characters widths between break point candidates need to be implemented then.

² See technical report version of this article for details.

Fig. 1. Network representing simple probabilistic line breaking algorithm.



$\{s_0, \dots, s_k\}$ maximizes $P(a, S, p, in)$ iff S maximizes

$$\sum_{j \in S^+} -(nlw_{s_j} - tw)^2 + \sum_{j \in S^-} (-(nlw_{s_j} - tw)^2 C_1 + C_2) - k.C_3 \quad (2)$$

where S^+ the breakpoints $j \in S$ such that $nlw_{s_j} > tw$, $S^- = S \setminus S^+$, $C_1 = \frac{\sigma^{+2}}{\sigma^{-2}}$, $C_2 = \sigma^{+2} \log \frac{\sigma^+}{\sigma^-}$, and $C_3 = \sigma^{+2} \log(2\pi\sigma^+)$. In words, maximizing the probability of acceptability $P(a, S, p, in)$ by choosing a set of break points S results in the same set of breakpoints as maximizing the sum over the lines longer than the text width tw (first sum term) and lines shorter than tw (second sum term). Comparing this with the formula that \TeX maximizes, equation (1), we see that \TeX uses the square of a sum of cubic functions, where we derived a quadratic one. Both have a term penalizing the number of lines (the term $-kC_3$ in (2), taking the reasonable assumption that $\log(2\pi\sigma^+) > 0$). Further, \TeX is concerned with the amount that spaces need to be expanded or squeezed, instead of looking at the whole line width. In Section 4 we will see that this does not matter too much.

Benefits of this simple probabilistic approach are that it allows for tuning with intuitively attractive parameters. Further, by choosing a smooth probability distribution for P^a with infinite domain, no threshold effects leading to overly long lines occur (as happens in \TeX 's approach, resulting in overfull hboxes). Finally, note that when instantiating the network with values of S , cw etc., the functional dependencies between nlw_i and nlw_{i-1} have no impact on the inference of a . So, we are left with a network that has a poly-tree structure for which finding the most likely configuration of b_i is linear in the size of the network [7].

3.2 Extending the probabilistic line breaking algorithm

The second assumption in the previous section can be relaxed to take interline effects in account. This allows discouraging multiple hyphenated lines in a row and preventing tight lines following loose lines. This information can be incorporated by defining new metrics and updating $P(S|p, in)$, the preference of a break-up.

There are two issues that need to be considered when factoring in more metrics. First, to ensure it is possible to find the break points S that are most likely to be accepted efficiently, the network structure that would model the newly created metrics should not create large cycles. It would be sufficient to ensure that the metrics are calculable for each character c_i from the information in slice i and $i - 1$ in the network. So when we have m metrics f_1, \dots, f_m , we can write $P(S, f_1, \dots, f_m|p) = \prod_{i=1}^n P(i \in S, f_{i,1}, \dots, f_{i,m}|p)$, where $f_{i,j}$ is the metric f_j calculated at character i . Second, a choice need to be made modeling the interoperations of the metrics on the probability of acceptability. A pragmatic choice is to assume that the metrics have independent impact on out preference for a break-up S at a particular character c_i . Then we can write $P(i \in S, f_{i,1}, \dots, f_{i,m}|p) = P(i \in S|p) \prod_{j=1}^m P(f_{i,j}|i \in S, p) \propto \prod_{j=1}^m P(f_{i,j}|i \in S, p)$.

Some metrics that could be calculated for each character are: tightness of previous line (one of 'tight', 'normal', 'loose'), number of previously hyphenated lines (one of 0, 1, 2, many), the amount spaces in a line need to be expanded to make the line fit the text width, and desirability of breakpoint (distinguishing desirability of spaces, hyphens, and hyphenation points). The latter requires the number of spaces at character i to be calculated as well.

A similar analysis as in the derivation of (2) reveals that each metric contributes as a separate sum, i.e., $\arg \max_S P(a, S, p, in) = \arg \max_S \sum_{j=1}^k m_{1,s_j} + \sum_{j=1}^k m_{2,s_j} + \dots + \sum_{j=1}^k m_{f,s_j}$. There is a similarity with (1) in that subtle effects impact additive. The difference is that \TeX 's penalties are difficult to interpret, especially because effects are multiplied (the term $p \cdot |p|$ in (1)).

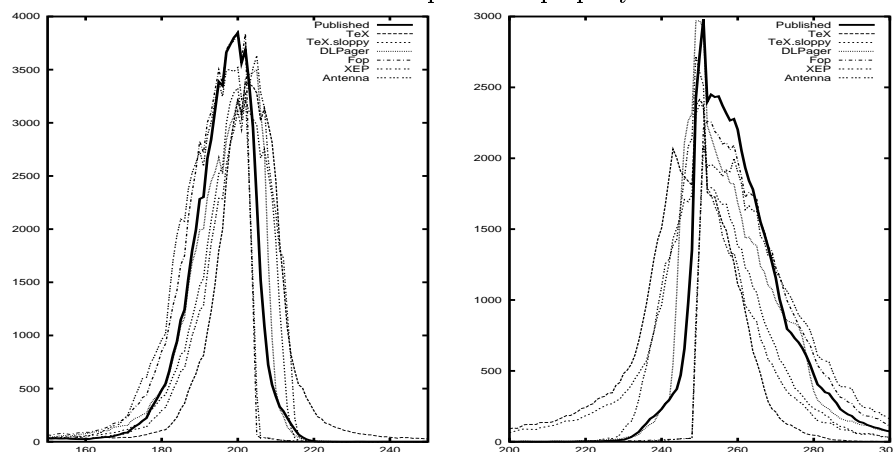
This extension has all the benefits of the simple probabilistic approach: intuitively tunable parameters, complexity linear in the number of character, and no threshold effects. Furthermore, it captures subtle interline effects and can be extended easily to capture even more factors.

4 Empirical results

To get an impression of the behavior of various line breaking algorithms, we looked at a set of existing publications and compared these with the output of some popular typesetting systems. Data was obtained from various scientific journals in the pharmaceutical area. Because the information was provided in PDF format, quite a bit of processing was required to get it into a useable format. Using the XPDF library³, text, font and location information was extracted from

³ Available from <http://www.foolabs.com/xpdf>.

Fig. 2. Natural line widths (left) and space widths (right) for published paragraphs and $\text{T}_{\text{E}}\text{X}$ and DLPager output. Left horizontal axis shows the line length in points (204pt is optimal), right the space width in 100th of a point (250 is optimal). Vertical axis shows the number of lines with the particular property.



the PDF files and strings glued together to lines, taking super and subscripts in account. Using simple pattern matching rules, any non-paragraph text, like headers and footers, titles, references etc., was removed and text lines glued to paragraphs. Because of tables and figures floating through the text, this was not 100% successful and a manual post-processing step was required to ensure broken paragraphs were glued or split. The database contained 5.459 paragraphs with 74.120 lines in total. Natural line widths were calculated taking kerning in account.

We compared the published paragraphs with $\text{T}_{\text{E}}\text{X}$ to establish two benchmarks. The $\text{T}_{\text{E}}\text{X}$ input was obtained by collecting text from each of the paragraphs, mapping special characters (like \leq , \pm , $\text{\textcircled{C}}$ etc) and generating a $\text{T}_{\text{E}}\text{X}$ file with a pagebreak after each paragraph. The $\text{T}_{\text{E}}\text{X}$ input uses the same font and paragraph settings as in the published paragraphs. Then \LaTeX was run to generate PDF and the same process as for published PDF was applied. However, this time paragraphs could be reliably extracted because each of them was output on its own page. $\text{T}_{\text{E}}\text{X}$ was run both with default settings and with sloppy paragraphs. The same process was repeated using DL Pager, a commercial composition engine from Datalogics⁴.

An informal experiment⁵ was performed where publication managers were asked to rank randomly selected paragraphs comparing published text with Pager and our algorithm. Paragraphs were presented side by side on a page to make comparison easier and the paragraphs were randomly ordered on the page. The result of this experiment was that publication managers tended to

⁴ See <http://www.datalogics.com> for details

⁵ See technical report version of this article for details of the experiment.

prefer the published result, with our new algorithm close by and Pager last. However, all three types of paragraphs were ranked first and last some of the time. Overall, this result is encouraging because it shows that Bayesian networks can actually help in increasing the typesetting quality.

An instant quantitative impression was obtained by measuring the distribution of the natural line widths and space widths of published material and lines generated by the typesetting systems. Though this does not show subtle inter-line behavior (such as appearance of multiple hyphenated lines), it still gives a good initial assessment of a system since looseness of lines is mentioned by publication managers as the primary indication of the quality of a line.

Figure 2 shows the distribution of the natural line widths for the published paragraphs and the \TeX output. Only middle lines are shown, not the first line (due to its indent) or last line. The ideal line is 204 points wide, if it is more, spaces have to be shrunk, or if it is less they have to be stretched. Figure 2 show clearly that handcrafted lines tend to be closer to the ideal length than \TeX lines. In fact default \TeX outputs a large number of lines that exceed the line length so much that there is not enough space in the line to shrink them, and the line exceeds the right boundary. Making paragraphs sloppy largely solves this problem, but there are even more looser lines than with the default setting. DL Pager output tends to have more lines closer to the ideal output compared to default \TeX . However, in cases where lines are loose, they tend to be very loose, making the tail on the left in the left side graph rather thick.

Also in Figure 2, natural line widths of three XSL-FO processors are shown. FOP version 0.20.4⁶ is an open source processor that uses a line breaking algorithm inspired by \TeX . XEP version 3.2⁷ and XSL Formatter version 2.4⁸ are commercial XSL-FO processors using unknown line breaking algorithms. All three XSL-FO processors use hyphenation rules inspired by \TeX 's hyphenation files, indicating that all of them use Liang's hyphenation algorithm [6]. The XSL-FO standard [8] leaves the implementation of the line breaking rules to the XSL-FO processor, within bounds of common sense line breaking rules as specified in the Unicode standard [2]. Consequently, there is very little a user can do to influence the layout of paragraphs when using XSL-FO. Figure 2 shows that all three processors are quite far from the published output. FOP and XSL-Formatter both are very reluctant to expand a space, as indicated by the sharp drop in the number of lines longer than the optimal 204 point width.

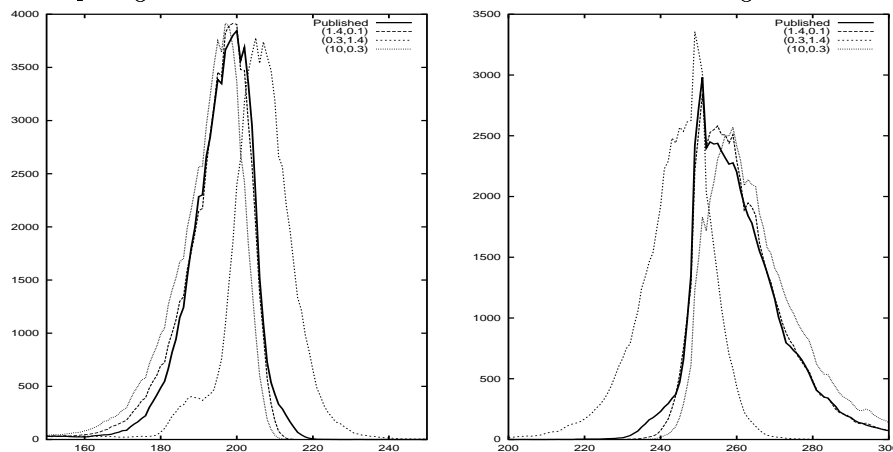
The right side of Figure 2 shows the distribution of space widths for the systems under investigation. Lines can be made a bit longer or shorter to justify by expanding or squeezing spaces. For example, in a line with 3 spaces and a natural line width of 201 point 1 point may be added to each of the spaces to expand the line to 204 point. The space width for a line is calculated as (natural line width - text width)/number of spaces where the paragraph width is 204 point and lines without spaces are ignored. The distribution for published text

⁶ Available from <http://xml.apache.org>.

⁷ Available from <http://www.renderx.com>.

⁸ Available from <http://www.antennahouse.com>.

Fig. 3. Natural line widths (left) and space widths (right) for published paragraphs and simple algorithm with various values of σ^- and σ^+ . Axis as in Figure 2.



is more peaked and has tails that are less fat than any of the systems. Sloppy \TeX has a wider distribution than plain \TeX and in fact any of the systems. The distribution of DLPager comes closest to the published results. Figure 2 illustrates once more the reluctance of Fop and Antenna House's tendency to squeeze spaces. XEP shows least hesitation in squeezing spaces of all the systems. Figure 2 shows that there is considerable difference in behavior and indicate that there is room for improvement in automatic line breaking.

Figure 3 shows the distribution of natural line widths (left) and spaces (right) of our probabilistic line breaking algorithm for different settings of σ^- and σ^+ . The published text is shown as reference. Interestingly, for $\sigma^- = 1.4$ and $\sigma^+ = 0.3$ we get a distribution that is especially close to the published text, closer than any of the other systems examined. By changing the values to $\sigma^- = 0.3$ and $\sigma^+ = 1.4$, we choose to accept tighter lines over loose lines and the line width distribution shifts to the right (so spaces get squeezed and its distribution goes to the left). Setting variances to $\sigma^- = 10$ and $\sigma^+ = 0.3$ gives a result left from the two. Remarkably, the probability of acceptability depends on the natural line widths only, not on the space width, like in \TeX 's algorithm. Figure 3 shows that this does not result in overly tight lines though.

We compared the simple probabilistic algorithm with the one extended with some of the terms of Section 3.2 by creating graphs similar to Figure 3. The graphs showed that the distributions of natural line widths and space widths are very close. Informal inspection of the output shows that the choice of break points indeed is influenced as desired by tuning parameters of the network.

5 Conclusions

We showed how a probabilistic interpretation of an ill defined problem, the problem of finding line breaks in a paragraph, can lead to an efficient new algorithm. The graphical model that results from the probabilistic interpretation has the advantage that it is easy to tune. Thanks to the architecture of the network structure being a polytree, the algorithm is linear in the number of characters in a paragraph. Line breaks are optimized taking the whole paragraph in account. We demonstrated how adding metrics to capture subtle typographical rules can be done without impacting the efficiency. The resulting algorithm does not show threshold effects (like T_EX) and it allows for easy incorporation of subtle typesetting properties. Empirical evidence suggests that this algorithm performs closer to results published through desk top publishing (DTP) than a number of existing systems.

There are a few limitations to our work. We only considered English text with US-English hyphenation rules in our experiments. However, different languages have other hyphenation rules and some even have different breaking rules, like east Asian languages [2]. Further, we only considered batch composition systems, but line breaking is routinely applied in word processors and DTP packages as well. These systems deal with incremental line breaking and the algorithm can be easily extended to deal with these limitations.

A bigger task is dealing with page breaks. It has been observed that page breaking can be interpreted as vertical version of the line breaking problem [1], so a probabilistic interpretation of this problem could be applied straightforwardly to find a new algorithm. An extra complication, however, is placement of floating objects, like figures, tables and footnotes. Also, there is the interaction between line breaks and paragraph breaks, making this a more challenging problem.

References

1. Jonathan Fine. Line breaking and page breaking. TUGBoat, vol 21 no 3, 210–221, 2000.
2. Asmus Freytag. Line Breaking Properties Unicode Standard Annex #14 (part of the Unicode Standard). Technical Report, 2002.
3. D.E. Knuth. Computers & Typesetting Volume A, The TeXbook. Reading, Massachusetts: Addison-Wesley, 1984.
4. D.E Knuth and M.F. Plass. Breaking Paragraphs into Lines. Software—Practice and Experience 11, p.1119–1184, 1981.
5. S.L. Lauritzen and D.J. Spiegelhalter. Local computations with probabilities on graphical structures and their applications to expert systems (with discussion). Journal of the Royal Statistical Society B, 50:157–224, 1988.
6. Franklin M. Liang. Word Hy-phen-a-tion by Com-put-er. Ph.D. Thesis, Department of Computer Science, Stanford University, August 1983.
7. J. Pearl. Probabilistic Reasoning in Intelligent Systems, Networks of Plausible Inference. Morgan Kaufmann, 1998.
8. Extensible Stylesheet Language (XSL). Version 1.0, W3C Recommendation, 15 October 2001.