Strategies against widows

Paul Isambert

Introduction

A widow line is the last line of a paragraph appearing as the first line of a page. Most typesetters try to avoid them when designing books (though some do not), whereas orphans (first line of a paragraph at the bottom of a page) and hyphenated bottom lines are often left untouched.

There are, to my knowledge, four different approaches to make widows enjoy marital life again.

1. The TEX approach. TEX avoids widows by assigning a penalty to them; when calculating the cost of a page, this penalty is taken into account and if there is a cheaper alternative, TEX will take it. This alternative in general involves stretching or shrinking space between paragraphs. This approach upsets any attempt at grid typesetting: the lines of a text might appear anywhere on the page, whereas a grid is normally used to display the flow of text.

This approach has two other drawbacks: first, it leads to ugly and unpredictable space between paragraphs (which is redundant); second, it is not sound: if breaking at a widow line is the cheapest alternative, then it won't be avoided. In a document made of text only, no widow is likely to be avoided, because there won't be enough stretchability.

2. Extra leading between lines. Another strategy is to increase the space between lines. Then the page has one line less than usual, which is given to the next page to accompany the widow. This is a very common practice in French books, and it upsets grid typesetting just like the previous approach. But at least it is less visible to the naked eye, and it cannot fail.

This approach is easily implemented in TEX. To make things work properly, let's first set the following:

```
\parskip=0pt
\clubpenalty=0 \brokenpenalty=0
\interlinepenalty=0
\vsize=31\baselineskip
\advance\vsize by \topskip
```

The first line removes any extra stretchability between paragraphs. The next two lines remove penalties associated with page breaking at an orphan, a hyphenated line, and a simple line (the latter is generally 0 by default anyway; it is used to invite TEX to break between rather than within paragraphs). The last part of the code sets the height of the textblock as a number of lines, instead of as an arbitrary length. The height of a line is `\baselineskip`, except the first line, whose height is `\topskip`. So here I've set a 32-line page. We'll use these values for all examples.

Extra leading between lines can be done in TEX because `\baselineskip`, as its name indicates, is a glue, not a dimension, hence it has stretchability and shrinkability. The idea is to set its stretchable part so that all interline glues can increase and fill the space left by the line moved to the next page. Suppose you have a page whose length is n lines. Then, in case there's a widow, you remove a line from the page, so that it contains $n-1$ lines, hence $n-2$ interline glues. The space to fill is one line, i.e. `\baselineskip`, so every glue should stretch by `\baselineskip`$/(n-2)$. Given our 32-line page, and assuming a baseline distance of 12pt, then:

```
\baselineskip=12pt plus .4pt
```

since $0.4 \times (32-2) = 12$.

(Note: `\baselineskip` might be confusing; its first part is not a glue at all, but the distance at which consecutive lines should be set; to do so, TEX inserts a glue item whose natural size depends on the depth of the line before and the height of the one after, but whose stretch and shrink components are the ones given to `\baselineskip`.)

Now, if `\widowpenalty` is larger than 100, then TEX will always increase space between lines to avoid a widow. This should also interact nicely with other rubber glues in the document. The value of 100 is no magical number; it's the badness of the page if TEX uses all the available stretch, which is the case at worst if there is just text on the page. Setting `\widowpenalty` to a larger value makes breaking at a widow an option with a higher cost, which therefore won't be taken.

A variant of this strategy decreases the space between lines in order to leave room for the widow. In our example, this makes a 33-line page, hence 32 interline glues, and the reader can check that it will be achieved with `\baselineskip=12pt minus .375pt`.

3. Lengthening or shortening the paragraph. The third approach, favored by French publishing houses with a typographic conscience, but which can lead to ugly results if done with a blind hand, redraws the offending paragraph so that it is one line longer or shorter. This strategy preserves the grid, but it can lead to paragraphs with large interword spacing in difficult cases.

In TEX this can be done automatically with the \looseness parameter. However this can easily fail, because the paragraph might not be lengthened or shortened. The following code inspects every paragraph, and tries to add or remove a line to paragraphs that would otherwise lead to a widow line. If it is impossible, an error message should be issued at output time, and the widow should be fixed by trying the same approach on a previous paragraph, this time by hand.

The idea is to build all paragraphs in a temporary box and check whether they have one more line than available on the page, leading to a widow. If so, redraw the paragraph with \looseness=1 or \looseness=-1. Here are our tools:

```
\chardef\linesperpage=32 \newbox\tempbox
\newcount\parheight \newcount\linesleft
\newcount\loosening
```

Note that we're interested in the number of lines of a paragraph modulo the number of lines per page. Indeed, if a paragraph is 45 lines long, and 12 lines remain on the current page, then it will fill that page and the next and leave a widow on the following one, just as a 13-line paragraph would. Thus we must consider a 45-line paragraph as a 13-line one. Hence the following macro, where \parheight is the height in lines of the paragraph under investigation:

```
\def\pagemodulo{%
  \ifnum\parheight>\linesperpage
    \advance\parheight by -\linesperpage
    \pagemodulo
  \fi}
```

We will proceed as follows. The \everypar token list contains a macro that stores the incoming paragraph. Meanwhile we also measure the remaining space on the page by subtracting \pagetotal (the length of the material already accumulated on the current page) from \pagegoal (the normal length of a page). However, there's a subtlety we must take into account. We'd assume that if, say, there are already three lines on the current page, then \pagetotal is \topskip (the height of the first line on any page) plus twice \baselineskip (the height of a normal line). But that is not the case, because the \parskip glue has been added (even though it is set to 0pt) between the previous paragraph and the current one, and TEX then considers that the material accumulated thus far has no depth, i.e. the depth of the last line (which would otherwise be recorded in \pagedepth) has been added to \pagetotal. Thus if three lines have been gathered, \pagetotal is \topskip plus twice \baselineskip plus the depth of the last line, which is fortunately recorded in \prevdepth. Hence the formula to compute the remaining number of lines on the page is:

$$\frac{\texttt{\pagegoal} - \texttt{\pagetotal} + \texttt{\prevdepth}}{\texttt{\baselineskip}}$$

The reader might ask, where has \topskip gone? It is neutralized when subtracting \pagetotal from \pagegoal. In case \pagegoal is null (and thus \topskip should be taken into account), then we don't need to compute anything; we know that the number of lines is \linesperpage.

So back to \everypar. Here's how it goes:

```
\everypar={%
  \ifdim\pagetotal=0pt
    \linesleft=\linesperpage
  \else
    \ifdim\pagetotal=\pagegoal
      \linesleft=\linesperpage
    \else
      \advance\linesleft by \pagegoal
      \advance\linesleft by -\pagetotal
      \divide\linesleft by \baselineskip
    \fi
  \fi
  \testpar}
```

The first part of the conditional is: if there's no material on the page, then there remains the full number of lines. The last part of the conditional is the computation described above. Since \everypar is inserted in horizontal mode, \prevdepth is not available any more; it was thus requested at the end of the previous paragraph (see below). Here, then, when we advance \linesleft, you should be aware that it's already \prevdepth in length. The middle part of the conditional might seem surprising. If \pagetotal is equal to \pagegoal, then the page is full, so we should be on a new page with \pagetotal set to 0pt, shouldn't we? No; if a page is full, then TEX has not decided yet that it is good; it takes an overfull page for TEX to decide that the page is filled, and reset \pagetotal.

Finally, \everypar launches \testpar, which first records in \parheight the number of lines of the paragraph and applies \pagemodulo. We also reset the value of \loosening, used below.

```
\def\testpar#1\par{%
  \setbox\tempbox=\vbox{%
    \everypar={}#1\endgraf
    \global\parheight=\prevgraf}%
  \pagemodulo \loosening=0
  % definition continues ...
```

Then we test whether the height of the paragraph is just one line more than \linesleft, which means a widow is going to happen. In this case, we rebuild the paragraph with \looseness set to -1 or 1, so as to remove or add a line. If the operation succeeds,

we set \loosening to the successful value; if not, we may send an error message (but it's simpler to leave this to the output routine).

```
\advance\linesleft by 1
\ifnum\parheight=\linesleft
  \setbox\tempbox=\vbox{%
    \everypar={}%
    \looseness=-1 #1\endgraf
    \ifnum\prevgraf<\parheight
      \global\loosening=-1
    \else
      \looseness=1 #1\endgraf
      \ifnum\prevgraf>\parheight
        \global\loosening=1
      \fi
    \fi}%
  \ifnum\loosening=0
    \errmessage{There'll be a widow!}%
  \fi
\fi
```

Finally, we release the paragraph with \looseness set to \loosening and record its depth in the \linesleft register for the next one:

```
  \looseness=\loosening
  #1\endgraf\linesleft=\prevdepth
} % end \testpar
```

This approach is very far from perfect. Above all, it isn't automatically successful. Or it may succeed, but not be the best solution. Indeed, it might sometimes be better to lengthen or shorten a previous paragraph, long enough so the operation is invisible. This strategy should be used with care anyway, and maybe care and automation are incompatible in this case.

(With less care still, one could manipulate the \tolerance value, or even interword space directly, so as to always succeed. But I won't spell it out, because this approach is already problematic enough if assignments are made within the paragraph — most of which could have been neutralized with the \globaldefs parameter, however —, so let's not trade typographic beauty any further.)

4. Adding or removing a line on the page. The final strategy is, according to Robert Bringhurst in his *Elements of Typographic Style,* used 'in most, if not all, the world's typographic cultures.' It consists in lengthening or shortening the page or pair of pages (the spread) by one line, so that the widow ends on the previous page or is given another line. I've never seen it done in French books, but all the American books I've inspected indeed do so.

Before going any further, I want to clarify the terminology. If a widow appears on page n, then we must be concerned with page $n-1$, and that's a bit confusing. So I will not talk about 'the widow on page 3' but rather 'the widow *from* page 2', because we have not the slightest interest in page 3. And I will say that page 2 *produces* a widow.

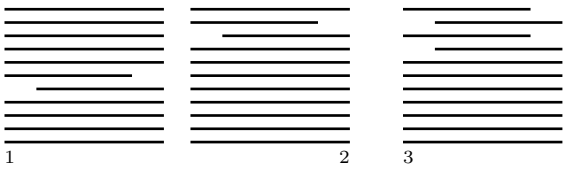

Fig. 1: A difficult case.

Adding or removing a line on a single page is quite simple. Victor Eijkhout gives example code on page 217 of *TEX by Topic.* However, books have a strong tendency to come in double pages, and treating pages one by one would ruin the design, because facing pages should have the same number of lines (except when one is the end of a chapter, of course). This means that we might redraw the left page to avoid a widow from the right one, and thus pages can't be shipped out at once even though they seem good. This also means that this approach won't always work. First, mending a page might make the other produce a widow. Second, when we try to avoid a widow from the right page, this might lead to another widow from the same page; indeed, if you add or remove lines on facing pages, the left page is affected by one line, but the right page suffers a two-line shift: one line because of the modification itself, and one line that went to or from the left page. And whereas a one-line change always fixes a widow, a two-line shift might produce one, if we're dealing with two-line paragraphs.

Figure 1 shows a spread and the left page overleaf that will lead to trouble. Supposing the road taken here is adding a line on facing pages, then removing a widow from the second page will make the left page produce one, and anyway since the second page will take two more lines from the following, as you can see another widow will appear. In this case, previous pages should be modified too, as we'll see at the end of the paper.

I've been talking about adding or removing lines as if both approaches could be used when needed, but you should stick to one or the other, otherwise differences between modified pages could be too easily spotted. However, Robert Bringhurst, for instance, consistently removes lines in the bulk of his book, but adds them in the appendix on foundries and in the bibliography (in both cases it also prevents last pages with only a few lines). But those two sections are distinct enough from the rest of the book to allow this strategic change: they aren't read like the main text to begin with. The

code below illustrates the removal approach, but adding lines would be the same thing with a couple of signs reversed here and there.

The strategy is a three-pass process in the output routine, where pass 3 only ships out the pages. If everything is ok in pass 1, we rebuild the pages with pass 3. Otherwise we rebuild them with pass 2, where \vsize has been decreased by \baselineskip; if it works here, we rebuild them with pass 3; if not, we also rebuild them with pass 3, but first reset \vsize to its normal value, and with an error message to signal that manual intervention is needed. Indeed, if the modification leads to nothing better, it is simpler to ship out the pages in an unmodified form so as to ease the appreciation of a modification on previous pages, to be done by hand.

Why can't we ship out the pages as soon as they are built in passes 1 or 2, provided they're good? Because of insertions, such as footnotes. For instance, suppose we're in pass 1, and the left page is ok; we can't ship it out, because we haven't examined the right page yet, and so we store it. Then comes the right page, which is assumed to be good too. So we could ship the pages, but what about footnotes? If there are any, it is impossible now to say what should appear on the left page and what on the right. Besides, if pages are bad, insertions should be put back in the stream with the pages themselves, because their splitting and/or position might change, and this is possible if and only if they aren't put in their boxes at output time, and thus can't be shipped either. I won't investigate this technical matter any further; to us it simply means that the \holdinginserts parameter should be set to a positive value for the first two passes and to 0 for pass 3.

Here we go. First, our tools:

```
\holdinginserts=1
\newbox\leftpage \newif\ifleftpage
\newif\iffirstpage \firstpagetrue
\newcount\passcount \passcount=1
```

Now, the redefinition of the output routine (in this and the following macros, all assignments are \global, because the output routine is executed in an implicit group):

```
\output{%
  \ifnum\passcount<3
    \ifnum\outputpenalty=\widowpenalty
      \global\advance\vsize by
        \ifnum\passcount=1 -\fi \baselineskip
      \global\advance\passcount by 1
      \ifnum\passcount=3
        \global\holdinginserts=0
      \fi
      \unboxpages
    \else
      \storepage
    \fi
  \else
    \ifnum\outputpenalty=\widowpenalty
      \errmessage{%
        Page \the\pageno\space produced a widow}%
    \fi
    \makeshipout
  \fi}
```

which reads as follows: if we're in pass 1 or 2 (i.e. \passcount < 3), and there is a widow (i.e. \outputpenalty $=$ \widowpenalty), then we modify \vsize by one baselineskip, either positively or negatively, depending on the pass we're in; in pass 1, we remove one \baselineskip, and in pass 2 we add it, thus returning to the default value of \vsize. Then we increase \passcount to prepare for the next pass, set \holdinginserts to 0 if we go to the last pass and put the page(s) constructed thus far back into the main vertical list with \unboxpages, explained below. If there was no widow, we simply store the current page with \storepage, which is also in charge of going to pass 3 for the shipout if both pages are built. Finally, if we are in pass 3, widows simply trigger an error message, and the page is shipped out anyway, with \makeshipout.

Next, the macros involved. First, \unboxpages:

```
\def\unboxpages{%
  \ifleftpage\else
    \iffirstpage\else
      \global\leftpagetrue
      \dimen0=\dp\leftpage
      \unvbox\leftpage
      \vskip\baselineskip
      \vskip-\topskip
      \vskip-\dimen0
  \fi \fi
  \unvbox255
  \ifnum\outputpenalty=10000 \penalty0
  \else \penalty\outputpenalty \fi}
```

This macro always puts box 255 back in the stream, because it's the current page; but if box 255 is the right page (i.e. \ifleftpage is false), then we must first release the left one, which was stored in the \leftpage box. However, there's one case when we're on a right page without a left page: if the page is the very first page of the document or chapter (\iffirstpage is true). In this case, we should simply release the current page. Now suppose that there is in fact a stored left page. Then we can't simply release it to in the stream, for the following reason: the interline glue that was first inserted between the last line of this

page and the first line of the next, so that their baselines are 12pt apart, has been discarded when the latter found its way to the top of box 255, where another glue was added, to match \topskip. Finally, TEX doesn't adjust interline spacing when lines are stacked with \unvbox, nor does it update \prevdepth. So we must do it by hand. The distance between the baseline of the bottom line of \leftpage and the baseline of the top line of box 255 should be \baselineskip. Part of this length is already filled by the depth of the bottom line, which is also the depth of the \leftpage box, and the height of the top line, which is \topskip. So we need a \vskip whose value is \baselineskip − \dp\leftpage − \topskip. Note that \dp\leftpage must be retrieved before the \unvbox, because the box is emptied there.

The unboxing of box 255 is much simpler. The insertion of a penalty is meant to balance the penalty of 10,000 that TEX always inserts in the main vertical list where it has broken a page. Thus this place again becomes an admissible breakpoint, which will be reused if pass 2 doesn't lead to better results and we rebuild the pages as they are now. We use the original penalty if there was one, because its exact value might be important (for instance for pass 3 to signal a widow).

The \storepage macro is very simple. Recall that it's executed by passes 1 and 2 when no widow is encountered. The left page is stored, whereas the completed right page launches pass 3.

```
\def\storepage{%
  \ifleftpage
    \global\leftpagefalse
    \global\setbox\leftpage=\box255
  \else
    \global\passcount=3 \global\holdinginserts=0
    {\setbox0=\vbox{\unvcopy255}%
    \ifdim\ht0=\topskip
      \ifnum\outputpenalty=-20000 \else
        \setbox0=\box255 \fi
    \fi}%
    \unboxpages
  \fi}
```

The part between braces deals with the end of the job, in case it happens on a left page. To put it simply, TEX is not happy because we have kept the left page in store when the job is supposed to terminate; so it adds an empty line to force the processing of the page, and this line might sometimes ends up at the top of the right page, thus creating a blank page. So we always analyze the right page, and if it's one line high and doesn't have the signature of a well-ended page (i.e. the \supereject penalty), then we delete it.

Finally, here is \makeshipout where headers, footers, and any other attributes of the final page should be added, and insertions placed; and, of course, the pages are shipped out. Here I show a simple page which contains only a page number. A very important point is that this page number can't be placed relative to the main text in box 255, because box 255 has a variable height whereas the page number should always be at the same place (if it were to go up and down, this approach would be a disaster). So suppose we want a default page (i.e. with an unmodified number of lines) where the page number is separated from the main text by a blank line; then we can't say

```
\shipout\vbox{%
  \box255 \vskip\baselineskip \pagenumber}
```

(where \pagenumber is supposed to produce the folio) for the reason above. Instead we must say:

```
\shipout\vbox to\totalpage{%
  \box255 \vfil \pagenumber}
```

where \totalpage = \vsize + 2 \baselineskip.

So, here's how it goes. We redundantly set \firstpagefalse on all shipouts, even though it matters only on the first one:

```
\newdimen\totalpage \totalpage=\vsize
\advance\totalpage by 2\baselineskip
\def\makeshipout{%
  \global\firstpagefalse
  \shipout\vbox to\totalpage{%
    \box255 \vfil
    \hbox to\hsize{%
      \ifleftpage \the\pageno\hfil
      \else \hfil\the\pageno \fi}%
    }%
  \advancepageno
```

The \advancepageno macro and \pageno counter are of course defined in plain TEX. In this example, the position of the page number depends on the evenness or oddness of the page, and is completely immaterial to what is at stake here (but it reminds us that we're doing all this because of facing pages).

Now, if we've just shipped out the right page, then pass 3 is over and we prepare for pass 1 again. We reset \vsize with \totalpage, so we don't have to store its original value anywhere. \csname page:\the\pageno\endcsname is a placeholder to be explained presently.

```
  \ifleftpage \global\leftpagefalse
  \else
    \global\leftpagetrue \global\passcount=1
    \global\holdinginserts=1 \vsize=\totalpage
    \global\advance\vsize by -2\baselineskip
    \csname page:\the\pageno\endcsname
  \fi
} % end \makeshipout
```

With this code, TEX will automatically remove widows whenever possible and flag them otherwise. In the latter case, we must be able to make a manual intervention on previous pages. Besides, this strategy is useful for page balancing in general. For instance, suppose (still on a 32-line page) that the left page is filled with 31 lines, and then comes a new section, which is separated from the preceding text by a blank line. Then the section title will end up on the right page and the left page will have only 31 lines. Suppose furthermore that the right page is completely filled, i.e. it has 32 lines. Then, even though the blank at the bottom of the left page is perfectly logical, it is generally better to remove a line from the right page so that the spread is balanced. This operation could be made automatically, but I will not investigate it here. I simply mention it as another case where the manual intervention can be used independently of widows.

The manual intervention is as follows: suppose that, say, page 35 produces a widow, and the automatic intervention fixes it, but creates a new one from page 34. Then the solution consists in removing lines on the previous spread instead, i.e. on pages 32 and 33. Then page 34 will shift by two lines instead of one, just like page 35, thus avoiding the widow (I leave it to the reader to check that this is indeed what happens). Of course, we might encounter harder situations, where we must modify both spreads, or still other spreads before, and so on and so forth. But the general idea remains the same: we must be able to indicate spreads where lines are to be removed. This is the meaning of the `\removeline{`⟨*pageno*⟩`}` macro. What `\removeline` does is to make the spread where ⟨*pageno*⟩ appears one line shorter, and go directly to pass 3 for this spread. If ⟨*pageno*⟩ is 1, then things are quite simple:

```
\def\removeline#1{%
  \bgroup \count0=#1
  \ifnum\count0=1
    \global\advance\vsize by -\baselineskip
    \global\passcount=3
    \global\holdinginserts=0
```

Otherwise, we build a macro with the number of the left page of the spread in its name. Its function is the same as above, i.e. prepare for pass 3 directly. And it is launched at the end of `\makeshipout` when page ⟨*pageno*⟩ − 1 has been shipped.

```
  \else
    \ifodd\count0 \advance\count0 by -1\fi
    \expandafter\gdef
      \csname page:\the\count0\endcsname{%
      \global\advance\vsize by -\baselineskip
      \global\passcount=3
      \global\holdinginserts=0
      }%
  \fi
  \egroup
} % end \removeline
```

Now we can specify, say, `\removeline{54}` or `\removeline{55}` at the beginning of the document so that the corresponding spread is made one line shorter, and this can be used for difficult widows, page balancing, and also to avoid short final pages by giving them additional lines (pages with only two or three lines of text are notoriously ugly).

Conclusion

The reader might have guessed that those four approaches have been ranked by order of (my) preference (even though approach 3 can lead to very good results when done very carefully). One may be surprised that TEX's default behavior is the worst . . . but actually this behavior is just an algorithm. In itself it is very good, but I believe it should be just a starting point, because it is unable to make meaningful decisions. For instance, although I'm not very fond of approach 2, I find it better than stretching space between paragraphs, because such space should be used to mark a logical pause, whereas extra leading between lines is unnoticeable (on a single page of course). And the difference between approaches 1 and 2 merely consists in moving the stretchable component from `\parskip` to `\baselineskip`, nothing fancier.

The third and fourth methods are harder but also lead to results not only better, but simply good. They investigate areas of TEX that are unfortunately not commonly studied, in part because the necessary underlying functionality (`\everypar`, `\output`) is appropriated by formats (which can't really do otherwise) and because they're supposed to be complex subjects. But the output routine is not harder to understand than `\expandafter`, quite the contrary, and it is worth it. Building a page is a process too important to be left to computer software, even TEX.

◇ Paul Isambert
Université de la Sorbonne Nouvelle
Paris 3
France
zappathustra (at) free dot fr