Zpravodaj Československého sdružení uživatelů TeXu

Max Chernoff

Automatically Removing Widows and Orphans with lua-widow-control

Zpravodaj Československého sdružení uživatelů TeXu, Vol. 32 (2022), No. 1-4, 49–76

Persistent URL: http://dml.cz/dmlcz/151108

Automatically Removing Widows and Orphans with lua-widow-control

Max Chernoff

The lua-widow-control package, for plain LuaTeX/LuaLaTeX/ConTeXt/OpTeX, removes widows and orphans without any user intervention. Using the power of LuaTeX, it does so without stretching any vertical glue or shortening any pages or columns. Instead, lua-widow-control automatically lengthens a paragraph on a page or column where a widow or orphan would otherwise occur.

To use the lua-widow-control package, all that most LaTeX users need do is place `\usepackage{lua-widow-control}` in their preamble. No further changes are required.

Keywords: LuaTeX, widows, orphans

1. Motivation

TeX provides top-notch typesetting: even 40 years after its first release, no other program produces higher quality mathematical typesetting, and its paragraph--breaking algorithm is still state-of-the-art. However, its page breaking is not quite as sophisticated as its paragraph breaking and thus suffers from some minor issues.

Unmodified TeX has only two familiar ways of dealing with widows and orphans: it can either shorten a page by one line, or it can stretch vertical whitespace. TeX was designed for mathematical and scientific typesetting, where a typical page has multiple section headings, tables, figures, and equations. For this style of document, TeX's default behaviour works quite well, since the slight stretching of whitespace between the various document elements is nearly imperceptible; however, for prose or other documents composed almost entirely of paragraphs, there is little vertical whitespace to stretch.

Since no ready-made, fully-automated solution to remove widows and orphans from all types of documents was available, I decided to create lua-widow-control.

2. What are widows and orphans?

2.1. Widows

A "widow" occurs when the majority of a paragraph is on one page or column, but the last line is on the following page or column. It not only looks quite odd

First published in *TUGboat* **43**:1 (Chernoff, 2022), pp. 28–39. Reprinted, with additions and corrections, with permission.

Widow	Orphan
A widow is when a paragraph's last line is placed on a different page than where it begins.	An orphan is when the first line of a paragraph occurs on the page before all the other lines.

Figure 1: The difference between widows and orphans. If we imagine that each box is a different page, then this roughly simulates how widows and orphans appear.

for a lone line to be at the start of the page, but it makes a paragraph harder to read since the separation of a paragraph and its last line disconnects the two, causing the reader to lose context for the widowed line.

2.2. Orphans

An "orphan" occurs when the first line of a paragraph is at the end of the page or column preceding the remainder of the paragraph. They are not as distracting for the reader, but they are still not ideal. Visually, widows and orphans are about equally disruptive; however, orphans tend not to decrease the legibility of a text as much as widows, so many authors choose to ignore them.

See Figure 1 for a visual reference.

2.3. Broken hyphens

"Broken" hyphens occur whenever a page break occurs in a hyphenated word. These are not related to widows and orphans; however, breaking a word across two pages is at least as disruptive for the reader as widows and orphans. TEX identifies broken hyphens in the same ways as widows and orphans, so lua-widow-control treats broken hyphens in the same way.

3. History and etymology

The concept of widows and orphans is nearly as old as printing itself. In *Mechanick exercises* (Moxon, 1683), a printers manual from 1683, we have:

> Nor do good *Compoſiters* account it good Workmanſhip to begin a *Page* with a *Break-line*, unleſs it be a very ſhort *Break*, and cannot be gotten in the foregoing *Page*; but if it be a long *Break*, he will let it be the *Direction-line* of the fore-going *Page*, and *Set* his *Direction* at the end of it. (p. 226)

However, the terms "widow" and "orphan" are much newer.

3.1. Widows

The earliest published source that I could find referencing "widows" in typography is *Webster's New International Dictionary* from 1934. However, no one—not even the editors of the dictionary (Brown, 1948a)—seems to know how it got there. Even then, the definition is somewhat different than it is now:

> widow, n. c. *Print.* A short line or single word carried over from the foot of one column or page to the head of a succeeding column or page. (Brown, 1948a)

Contrast this with the modern definition:

> The stub-ends left when paragraphs end on the first line of a page are called widows. They have a past but not a future, and they look foreshortened and forlorn. (Bringhurst, 2004)

which includes a single lone line of any length.

3.2. Orphans

The term "orphan" is even more confusing. Its initial usage seems to have occurred some time after "widow" (Brown, 1948a), and it is given many contradictory definitions. Most sources define an orphan as a first line at the bottom of the page and a widow as the last line at the top (Bringhurst, 2004; Brown, 1948a; Brown, 1948b; Isambert, 2010; Knuth, 2021; Mittelbach, 2021; Oxford English Dictionary, 2021b; Oxford English Dictionary, 2021c); however, some sources define these two terms as *exact opposites* of each other, with a widow as a first line at the bottom of the page and an orphan as the last line! (Ambrose; Harris, 2007; Brown, 1948a; Hunt, 2020; Oxford English Dictionary, 2021b; Saltz, 2019) This usage is plain wrong; nevertheless, it is sufficiently common that you need to be careful when you see the terms "widow" and "orphan".

Similarly to the term "widow", *The Elements of Typographic Style* (Bringhurst, 2004) provides a succinct definition of the term "club", along with a helpful mnemonic:

> Isolated lines created when paragraphs begin on the last line of a page are known as orphans. They have no past, but they do have a future. (Bringhurst, 2004)

3.3. Clubs

The TEXbook never refers to "orphans" as such; rather, it refers to them as "clubs". This term is remarkably rare: I could only find a *single* source published before *The TEXbook*—a compilation article about the definition of "widow"—that mentions a "club line":

> The Dictionary staff informs me that they have no example of the use of the word widow in the typographical sense. […]

> Mr. Watson of the technical staff says that the Edinburgh printing houses referred to it as a "clubline". (Brown, 1948a, p. 4)

> To my knowledge, a 'widow', or 'widow-line,' is a short line, forming the end of a paragraph, which is carried over from the foot of a page or column to the top of the succeeding one. […]
>
> To my personal knowledge, in typographical parlance in Edinburgh, Scotland, the 'widow' is called a 'club-line.' (Brown, 1948a, p. 23)

Both quotes above are from separate authors, and they each define a "club" like we define "widow", not an "orphan". In addition, they both mention that the term is only used in Scotland. Even the extensive OED—which lists 17 full definitions and 103 subdefinitions for the noun "club"—doesn't recognize the phrase. (Oxford English Dictionary, 2021a)

I spent a few hours searching through Google Books and my university library catalogue, but I could not find a single additional source. However, Don Knuth—the creator of TEX—read the original article (Chernoff, 2022) and sent me this reply:

> I cannot remember where I found the term "club line". Evidently the books that I scoured in 1977 and 1978 had taught me only that an isolated line, caused by breaking between pages in the midst of a paragraph, was called a "widow"; hence TEX78 had only "`\chpar4`" to change the "`widowpenalty`". Sometime between then and TEX82 I must have come across what appeared to be an authoritative source that distinguished between widows at the beginning of a paragraph and orphans or club lines at the end. I may have felt that the term "orphan" was somewhat pejorative, who knows?[1]

So this (somewhat) resolves the question of where the term "club" came from.

4. Pagination in TEX

Let's move on to looking at how TEX treats these widows and orphans.

4.1. Algorithm

It is tricky to understand how lua-widow-control works if you aren't familiar with how TEX breaks pages and columns. For a full description, you should consult Chapter 15 of *The TEXbook* (Knuth, 2021) ("How TEX Makes Lines into Pages");

[1] Note that this definition is somewhat mistaken. Widows are located either at the *end* of a paragraph, or the beginning of a *page or column*. Likewise, orphans/clubs appear at the *beginning* of a paragraph or at the end of a *page or column*.

however, this goes into much more detail than most users require, so here is a *very* simplified summary of TEX's page breaking algorithm:

TEX fills the page with lines and other objects until the next object will no longer fit. Once no more objects will fit, TEX will align the bottom of the last line with the bottom of the page by stretching any available vertical spaces if (in LATEX) `\flushbottom` is set; otherwise, it will break the page and leave the bottom empty.

However, some objects have penalties attached. Penalties encourage or discourage page breaks from occurring at specific places. For example, LATEX sets a negative penalty before section headings to encourage a page break there; conversely, it sets a positive penalty after section headings to discourage breaking.

To reduce widows and orphans, TEX sets weakly-positive penalties between the first and second lines of a paragraph to prevent orphans, and between the penultimate and final lines to prevent widows.

One important note: once TEX begins breaking a page, it never goes back to modify any content on the page. Page breaking is a localized algorithm, without any backtracking.

4.2. Behaviour

Merely describing the algorithm doesn't allow us to intuitively understand how TEX deals with widows and orphans.

Due to the penalties attached to widows and orphans, TEX tries to avoid them. Widows and orphans with small penalties attached—like LATEX's default values of 150—are only lightly coupled to the rest of the paragraph, while widows and orphans with large penalties—values of 10 000 or more—are treated as infinitely bad and are thus unbreakable. Intermediate values behave just as you would expect, discouraging page breaks proportional to their value.

When TEX goes to break a page, it tries to avoid breaking at a location with a high penalty. How it does so depends on a few settings:

4.2.1. `\flushbottom` and `\normalbottom`

With the settings `\normalbottom` (Plain TEX) or `\flushbottom` (LATEX), TEX is willing to stretch any glue on the page by an amount roughly commensurate to the magnitude of the penalty: for small `\clubpenalty` and `\widowpenalty` values, TEX will only slightly stretch the glue on the page before creating a widow or orphan; for very large penalties, TEX will stretch the glue by a near-infinite amount.

This corresponds to the "Stretch" column in Figure 2. It is the default behaviour of Plain TEX, and of the standard LATEX classes when the `twocolumn` option is given.

4.2.2. `\raggedbottom`
When `\raggedbottom` is set, TEX won't stretch any glue. Instead, for sufficiently high `\clubpenalty` and `\widowpenalty` values, TEX will shorten the page or column by one line in order to prevent the widow or orphan from being created.

This corresponds to the "Shorten" column in Figure 2 and is the default behaviour of the LATEX classes when the `twocolumn` option is not given.

5. `\looseness`

Before we can continue further, we need to discuss one more TEX command: `\looseness`. The following is excerpted from Chapter 14 of *The TEXbook* (Knuth, 2021) ("How TEX Breaks Paragraphs into Lines"):

> If you set `\looseness=1`, TEX will try to make the current paragraph one line longer than its optimum length, provided that there is a way to choose such breakpoints without exceeding the tolerance you have specified for the badnesses of individual lines. Similarly, if you set `\looseness=2`, TEX will try to make the paragraph two lines longer; and `\looseness=-1` causes an attempt to make it shorter. [...]
>
> For example, you can set `\looseness=1` if you want to avoid a lonely "club line" or "widow line" on some page that does not have sufficiently flexible glue, or if you want the total number of lines in some two-column document to come out to be an even number.
>
> It's usually best to choose a paragraph that is already pretty "full", i.e., one whose last line doesn't have much white space, since such paragraphs can generally be loosened without much harm. You might also want to insert a tie between the last two words of that paragraph, so that the loosened version will not end with only one "widow word" on the orphans line; this tie will cover your tracks, so that people will find it hard to detect the fact that you have tampered with the spacing. On the other hand, TEX can take almost any sufficiently long paragraph and stretch it a bit, without substantial harm.

The widow and orphan removal strategy suggested in the second paragraph works quite well; however, it requires manual editing each and every time a page or paragraph is rewritten or repositioned.

6. Alternate removal strategies

There have been a few previous attempts to improve upon TEX's previously discussed widow and orphan-handling abilities; however, none of these have been

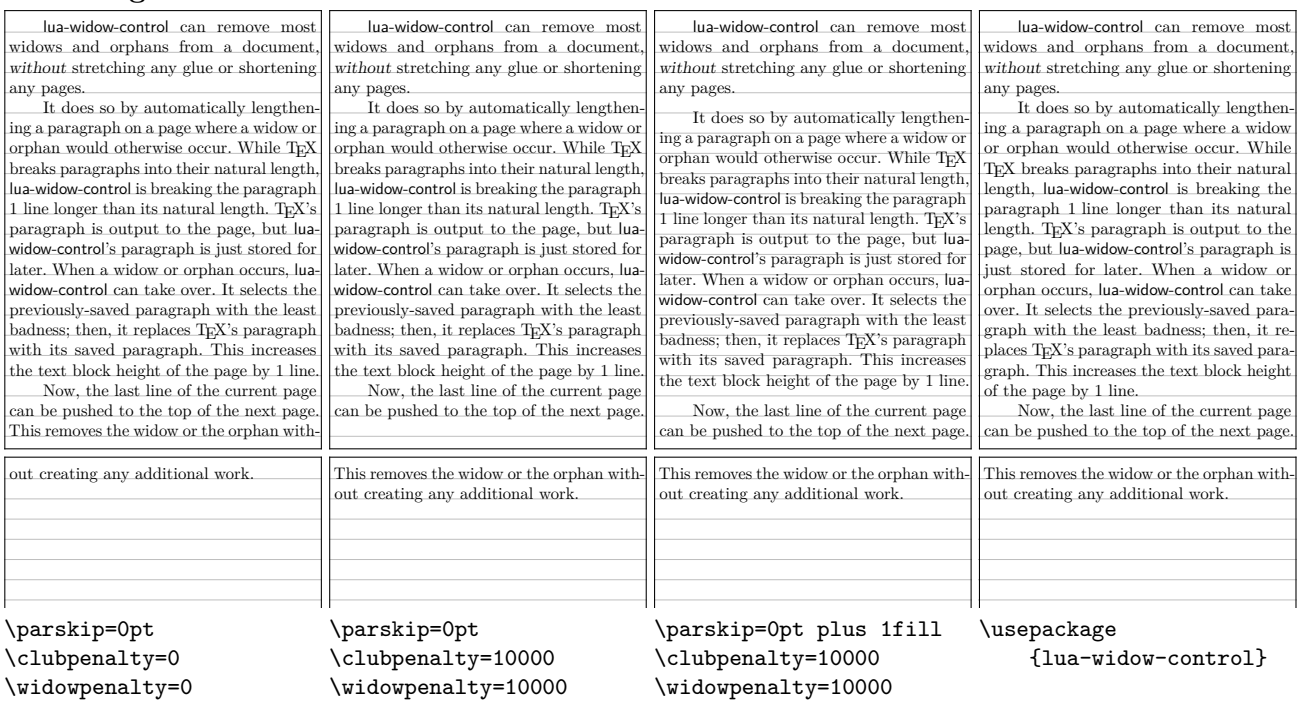

Figure 2: A visual comparison of various automated widow-handling techniques.

able to automatically remove widows and orphans without stretching any vertical glue or shortening any pages.

The articles "Strategies against widows" (Isambert, 2010) and "Managing forlorn paragraph lines (a.k.a. widows and orphans) in LaTeX" (Mittelbach, 2018b) both begin with comprehensive discussions of the methods of preventing widows and orphans. They agree that widows and orphans are bad and ought to be avoided; however, they differ in their solutions. *Strategies* proposes an output routine that reduces the length of facing pages by one line when necessary to remove widows and orphans, while *Managing* proposes that the author manually rewrites or adjusts `\looseness` when needed.

The post *Paragraph callback to help with widows/orphans hand tuning* (jeremie, 2017) contains a file `widow-assist.lua` that automatically detects which paragraphs can be safely shortened or lengthened by one line. The widows-and-orphans package (Mittelbach, 2021) alerts the author to the pages that contain widows or orphans. Combined, these packages make it simple for the author to quickly remove widows and orphans by adjusting the values of `\looseness`; however, it still requires the author to make manual source changes after each revision.

Another article suggests a fully-automated solution to remove widows and orphans (Mittelbach, 2018a). This would seem to offer a complete solution; however, it requires multiple passes, an external tool, and has not yet been publicly released.

lua-widow-control is essentially a combination of `widow-assist.lua` (jeremie, 2017) and widows-and-orphans (Mittelbach, 2021), although its implementation is independent of both: when the `\outputpenalty` value indicates that a widow or orphan has occurred, Lua is used to find a stretchable paragraph. What lua-widow-control mainly adds on top of these packages is automation: it eliminates the requirement for any manual adjustments or changes to your document's source.

7. Visual comparison

Although TeX's page breaking algorithm is reasonably straightforward, it can lead to complex behaviour when widows and orphans are involved. The usual choices, when rewriting is not possible, are to ignore them, stretch some glue, or shorten the page. Figure 2 has a visual comparison of these options, which we'll discuss in the following:

7.1. Ignore

As you can see, the last line of the page is on a separate page from the rest of its paragraph, creating a widow. This is usually highly distracting for the reader, so it is best avoided for the reasons previously discussed.

7.2. Shorten

This page did not leave any widows, but it did shorten the previous page by one line. Sometimes this is acceptable, but usually it looks bad because pages will then have different text-block heights. This can make the pages look quite uneven, especially when typesetting with columns or in a book with facing pages.

7.3. Stretch

This page also has no widows and it has a flush bottom margin. However, the space between each pair of paragraphs had to be stretched.

If this page had many equations, headings, and other elements with natural space between them, the stretched out space would be much less noticeable. TEX was designed for mathematical typesetting, so it makes sense that this is its default behaviour. However, in a page with mostly text, these paragraph gaps look unsightly.

Also, this method is incompatible with grid typesetting, where all vertical glue stretching must be quantised to the height of a line.

7.4. lua-widow-control

lua-widow-control has none of these issues: it eliminates the widows in a document while keeping a flush bottom margin and constant paragraph spacing.

To do so, lua-widow-control lengthened the second paragraph in Figure 2 by one line. If you look closely, you can see that this stretched the interword spaces. This stretching is noticeable when typesetting in a narrow text block, but is mostly imperceptible with larger widths.

lua-widow-control automatically finds the "best" paragraph to stretch, so the increase in interword spaces should almost always be minimal.

8. Installation and standard usage

The lua-widow-control package was first released in October 2021. It is available in the default installations of both MiKTEX and TEX Live, although you will need recent versions of either.

You may also download lua-widow-control manually from either CTAN,[2] the ConTEXt Garden,[3] or GitHub,[4] although it is best if you can install it through your TEX distribution.

[2] ctan.org/pkg/lua-widow-control

[3] modules.contextgarden.net/cgi-bin/module.cgi/action=view/id=127

[4] github.com/gucci-on-fleek/lua-widow-control/releases/latest/

As its name may suggest, lua-widow-control *requires* LuaTEX or LuaMetaTEX regardless of the format used. With that in mind, using lua-widow-control is quite simple:

Plain TEX	`\input lua-widow-control`
OpTEX	`\load[lua-widow-control]`
LATEX	`\usepackage{lua-widow-control}`
ConTEXt	`\usemodule[lua-widow-control]`

And that's usually enough. Most users won't need to do anything else since lua-widow-control comes preconfigured and ready-to-go.

9. Options

Nevertheless, lua-widow-control does have a few options.

lua-widow-control tries very hard to have a "natural" user interface with each format, so how you set an option heavily depends on how you are running lua-widow-control. Also note that not every option is available in every format.

Some general guidelines:

Plain TEX/OpTEX	Specially-named `\lwc`⟨*option*⟩ commands and registers are provided for all options.
LATEX	Options can be set either as package options or at any point in the document with `\lwcsetup`.
ConTEXt	Always use `\setuplwc`.

9.1. Disabling

You may want to disable lua-widow-control for certain portions of your document. You can do so with the following commands:

Plain TEX/OpTEX	`\lwcdisable`
LATEX	`\lwcsetup{disable}`
ConTEXt	`\setuplwc[state=stop]`

This prevents lua-widow-control from stretching any paragraphs that follow. If a page has earlier paragraphs where lua-widow-control was still enabled and a widow or orphan is detected, lua-widow-control will still attempt to remove the widow or orphan.

9.2. Enabling

lua-widow-control is enabled as soon as the package is loaded. If you have previously disabled it, you will need to re-enable it to save new paragraphs.

Plain TEX/OpTEX	`\lwcenable`
LATEX	`\lwcsetup{enable}`
ConTEXt	`\setuplwc[state=start]`

9.3. Automatically disabling

You may want to disable lua-widow-control for certain commands where stretching is undesirable such as section headings. Of course, manually disabling and then enabling lua-widow-control multiple times throughout a document would quickly become tedious, so lua-widow-control provides some options to do this automatically for you.

lua-widow-control automatically patches the default LATEX, ConTEXt, Plain TEX, OpTEX, memoir, KOMA-Script, and titlesec section commands, so you don't need to patch these. Any others, though, you'll need to patch yourself.

Plain TEX/OpTEX	`\lwcdisablecmd{`⟨`\`*macro*⟩`}`
LATEX	`\lwcsetup{disablecmds={`⟨*csnameone*⟩`,`⟨*csnametwo*⟩`}}`
ConTEXt	`\prependtoks\lwc@patch@pre\to\everybefore`⟨*hook*⟩ `\prependtoks\lwc@patch@post\to\everyafter`⟨*hook*⟩

9.4. `\emergencystretch`

lua-widow-control defaults to an `\emergencystretch` value of 3 em for stretched paragraphs, but you can configure this.

lua-widow-control will only use the `\emergencystretch` when it cannot extend a paragraph in any other way, so it is fairly safe to set this to a large value. TEX accumulates badness when `\emergencystretch` is used (Knuth, 1989), so it's pretty rare that a paragraph that requires any `\emergencystretch` will actually be used on the page.

Plain TEX/OpTEX	`\lwcemergencystretch=`⟨*dimension*⟩
LATEX	`\lwcsetup{emergencystretch=`⟨*dimension*⟩`}`
ConTEXt	`\setuplwc[emergencystretch=`⟨*dimension*⟩`]`

9.5. Penalties

You can also manually adjust the penalties that TEX assigns to widows and orphans. Usually, the defaults are fine, but there are a few circumstances where you may want to change them.

Plain TEX/OpTEX	`\widowpenalty=`⟨*integer*⟩
	`\clubpenalty=`⟨*integer*⟩
	`\brokenpenalty=`⟨*integer*⟩
LATEX	`\lwcsetup{ widowpenalty=`⟨*integer*⟩`}`
	`\lwcsetup{orphanpenalty=`⟨*integer*⟩`}`
	`\lwcsetup{brokenpenalty=`⟨*integer*⟩`}`
ConTEXt	`\setuplwc[ widowpenalty=`⟨*integer*⟩`]`
	`\setuplwc[orphanpenalty=`⟨*integer*⟩`]`
	`\setuplwc[brokenpenalty=`⟨*integer*⟩`]`

The value of these penalties determines how much TEX should attempt to stretch glue before passing the widow or orphan to lua-widow-control. If you set the values to 1 (default), TEX will stretch nothing and immediately trigger lua-widow-control; if you set the values to 10 000, TEX will stretch infinitely and lua-widow-control will never be triggered. If you set the value to some intermediate number, TEX will first attempt to stretch some glue to remove the widow or orphan; only if it fails will lua-widow-control come in and lengthen a paragraph. As a special case, if you set the values to 0, both TEX and lua-widow-control will completely ignore the widow or orphan.

lua-widow-control will pick up on the values of `\widowpenalty`, `\clubpenalty`, and `\brokenpenalty` regardless of how you set them, so the use of these dedicated keys is entirely optional.

9.6. `\nobreak` behaviour

When lua-widow-control encounters an orphan, it removes it by moving the orphaned line to the next page. The majority of the time, this is an appropriate solution. However, if the orphan is immediately preceded by a section heading (or `\nobreak`/`\penalty 10000`), lua-widow-control would naïvely separate a section heading from the paragraph that follows. This is almost always undesirable, so lua-widow-control provides some options to configure this.

Plain TEX/OpTEX	`\lwcnobreak{`⟨*value*⟩`}`
LATEX	`\lwcsetup{nobreak=`⟨*value*⟩`}`
ConTEXt	`\setuplwc[nobreak=`⟨*value*⟩`]`

The default value, `keep`, *keep*s the section heading with the orphan by moving both to the next page. The advantage to this option is that it removes the orphan and retains any `\nobreak`s; the disadvantage is that moving the section heading can create a large blank space at the end of the page. The value `split` *split*s up the section heading and the orphan by moving the orphan to the next page while

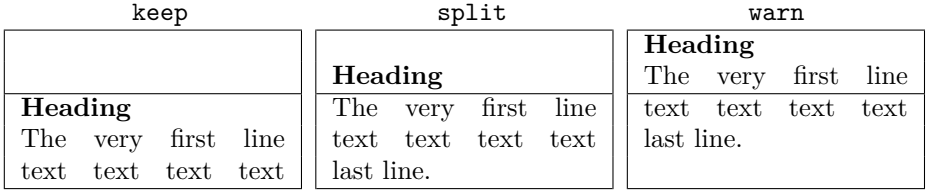

Figure 3: A visual comparison of the `nobreak` option values.

leaving the heading behind. This is usually a bad idea, but exists for the sake of flexibility. The value `warn` causes lua-widow-control to give up on the page and do nothing, leaving an orphaned line. lua-widow-control *warns* the user so that they can manually remove the orphan.

See Figure 3 for a visual reference.

9.7. Maximum cost

lua-widow-control ranks each paragraph on the page by how much it would "cost" to lengthen that paragraph. By default, lua-widow-control selects the paragraph on the page with the lowest cost; however, you can configure it to only select paragraphs below a selected cost.

If there aren't any paragraphs below the set threshold, then lua-widow-control won't remove the widow or orphan and will instead issue a warning.

Plain TEX/OpTEX	`\lwcmaxcost=`⟨*integer*⟩
LATEX	`\lwcsetup{max-cost=`⟨*integer*⟩`}`
ConTEXt	`\setuplwc[maxcost=`⟨*integer*⟩`]`

Based on my testing, `max-cost` values less than 1 000 cause completely imperceptible changes in interword spacing; values less than 5 000 are only noticeable if you are specifically trying to pick out the expanded paragraph on the page; values less than 15 000 are typically acceptable; and larger values may become distracting. lua-widow-control defaults to an infinite `max-cost`, although the "strict" and "balanced" modes sets the values to 5 000 and 10 000, respectively.

9.8. Draft mode

lua-widow-control has a "draft mode" which shows how lua-widow-control processes pages.

Plain TEX/OpTEX	`\lwcdraft 1`
LATEX	`\lwcsetup{draft}`
ConTEXt	`\setuplwc[draft=start]`

The draft mode has two main features:

First, it colours lines in the document according to their status. Any remaining widows and orphans will be coloured red, any expanded paragraphs will be coloured green, and any lines moved to the next page will be coloured blue.

Second, this draft mode shows the paragraph costs at the end of each paragraph, in the margin.

This draft mode leads to a neat trick: if you don't quite trust lua-widow-control, or you're writing a document whose final version will need to be compilable by both pdfLaTeX and LuaLaTeX, you can load the package with:

```
\usepackage[draft, disable]{lua-widow-control}
```

This way, all the widows and orphans will be coloured red and listed in your log file. When you go through the document to try and manually remove the widows and orphans—whether through the `\looseness` trick or by rewriting certain lines—you can easily find the best paragraphs to modify by looking at the paragraph costs in the margins. If you're less cautious, you can compile your document with lua-widow-control enabled as normal and inspect all the green paragraphs to see if they look acceptable to you.

You can also toggle the paragraph colouring and the cost displays individually:

Plain TeX/OpTeX	`\lwcshowcosts 1`
	`\lwcshowcolours 0`
LaTeX	`\lwcsetup{showcosts=true}`
	`\lwcsetup{showcolours=false}`
ConTeXt	`\setuplwc[showcosts=start]`
	`\setuplwc[showcolours=stop]`

10. Presets

As you can see, lua-widow-control provides quite a few options. Luckily, there are a few presets that you can use to set multiple options at once. These presets are a good starting point for most documents, and you can always manually override individual options.

These presets are only available for LaTeX and ConTeXt.

LaTeX	`\lwcsetup{`⟨*preset*⟩`}`
ConTeXt	`\setuplwc[`⟨*preset*⟩`]`

10.1. `default`

If you use lua-widow-control without any options, it defaults to this preset. In default mode, lua-widow-control takes all possible measures to remove widows

and orphans and will not attempt to stretch any vertical glue. This usually removes $>95\%$ of all possible widows and orphans. The catch here is that this mode is quite aggressive, so it often leaves behind some fairly "spacey" paragraphs.

This mode is good if you want to remove (nearly) all widows and orphans from your document, without fine-tuning the results.

10.2. `strict`

lua-widow-control also offers a strict mode. This greatly restricts lua-widow-control's tolerance and makes it so that it will only lengthen paragraphs where the change will be imperceptible.

The caveat with strict mode is that—depending on the document—lua-widow-control will be able to remove less than a third of the widows and orphans. For the widows and orphans that can't be automatically removed, a warning will be printed to your terminal and log file so that a human can manually fix the situation.

This mode is good if you want the best possible typesetting and are willing to do some manual editing.

10.3. `balanced`

Balanced mode sits somewhere between default mode and strict mode. This mode first lets TEX stretch a little glue to remove the widow or orphan; only if that fails will it then trigger lua-widow-control. Even then, the maximum paragraph cost is capped. Here, lua-widow-control can usually remove 90% of a document's potential widows and orphans, and it does so while making a minimal visual impact.

This mode is recommended for most users who care about their document's typography. This mode is not the default since it doesn't remove all widows and orphans: it still requires a little manual intervention.

11. Compatibility

The lua-widow-control implementation is almost entirely in Lua, with only a minimal TEX footprint. It doesn't modify the output routine or `\everypar` and it doesn't insert any whatsits. This means that it should be compatible with nearly any TEX package, class, and format. Most changes that lua-widow-control makes are not observable on the TEX side.

However, on the Lua side, lua-widow-control modifies much of a page's internal structure. This should not affect any TEX code; however, it may surprise Lua code that modifies or depends on the page's low-level structure. This does not affect Plain TEX or LATEX where even most Lua-based packages don't depend on

Table 2: lua-widow-control options set by each mode.

Option	default	balanced	strict
max-cost	∞	10000	5000
emergencystretch	3em	1em	0pt
nobreak	keep	keep	warn
widowpenalty	1	500	1
orphanpenalty	1	500	1
brokenpenalty	1	500	1

the node list structure. ConTEXt *does* depend on this internal node structure; however, I have carefully tested the package to ensure that this causes no issues.

Finally, keep in mind that adding lua-widow-control to a document will almost certainly change its page break locations.

11.1. Formats

lua-widow-control runs on all known LuaTEX-based formats: Plain LuaTEX, LuaLATEX, ConTEXt MkIV, and OpTEX. Unless otherwise documented, all features should work equally well in all formats.

lua-widow-control is also fully-compatible with the LuaMetaTEX-based formats: ConTEXt MkXL/LMTX, LuaMetaLATEX, and LuaMetaPlain (Krüger, 2022). ConTEXt MkXL works equally well as ConTEXt MkIV and LuaLATEX; however, LuaMetaLATEX and LuaMetaPlain support is still quite early. All features should work, although there are still a few minor bugs.

All told, lua-widow-control supports 7 different format/engine combinations.

11.2. Columns

Since TEX and the formats implement column breaking and page breaking through the same internal mechanisms, lua-widow-control removes widows and orphans between columns just as it does with widows and orphans between pages.

lua-widow-control is known to work with the LATEX class option `twocolumn` and the two-column output routine from Chapter 23 of *The TEXbook* (Knuth, 2021).

11.3. Performance

lua-widow-control runs entirely in a single pass, without depending on any `.aux` files or the like. Thus, it shouldn't meaningfully increase compile times. Although

lua-widow-control internally breaks each paragraph twice, modern computers break paragraphs near-instantaneously, so you are not likely to notice any slowdown.

lua-widow-control has been carefully tested to ensure that there are no memory leaks, so lua-widow-control can now easily compile documents $> 10\,000$ pages long.

11.4. ε-TEX penalties

Knuth's original TEX has three basic line penalties: `\interlinepenalty`, which is inserted between all lines; `\clubpenalty`, which is inserted after the first line; and `\widowpenalty`, which is inserted before the last line. The ε-TEX extensions (The $\mathcal{NTS}$ Team, 1998) generalize these commands with a syntax similar to `\parshape`: with `\widowpenalties` you can set the penalty between the last, second last, and nth last lines of a paragraph; `\interlinepenalties` and `\clubpenalties` behave similarly.

The lua-widow-control package makes no explicit attempts to support these new `-penalties` commands. Specifically, if you give a line a penalty that matches either `\widowpenalty` or `\clubpenalty`, lua-widow-control will treat the lines exactly as it would a widow or orphan. So while these commands won't break lua-widow-control, they are likely to lead to some unexpected behaviour.

12. Short last lines

When lengthening a paragraph with `\looseness`, it is common advice to insert ties (`~`) between the last few words of the paragraph to avoid overly-short last lines (Knuth, 2021). lua-widow-control does this automatically, but instead of using ties or `\hbox`es, it uses the `\parfillskip` parameter (Knuth, 2021; Wermuth, 2018; Olšák, 1997). When lengthening a paragraph (and only when lengthening a paragraph—remember, lua-widow-control doesn't interfere with TEX's output unless it detects a widow or orphan), lua-widow-control sets `\parfillskip` to `0.75\hsize plus 0.05\hsize minus 0.75\hsize`. This normally makes the last line of a paragraph be at least 20% of the overall paragraph's width, thus preventing ultra-short lines.

13. How it works

lua-widow-control uses a fairly simple algorithm to eliminate widows and orphans, but there are a few subtleties.

13.1. Setup

lua-widow-control sets the `\clubpenalty`, `\widowpenalty`, and `\brokenpenalty` parameters to sentinel values of 1. This will signal to lua-widow-control when a widow or orphan occurs, yet it is small enough that it won't stretch any glue.

lua-widow-control also enables LuaTEX's microtypographic extensions (Thành, 2001). This isn't strictly necessary; however, it significantly increases the number of paragraphs that can be acceptably "loosened".

That is all that happens on the TEX end. The rest of lua-widow-control is pure Lua.

13.2. Paragraph breaking

First, lua-widow-control hooks into the paragraph breaking process, before any output routines or page breaking.

Before a paragraph is broken by TEX, lua-widow-control grabs the unbroken paragraph. Then lua-widow-control breaks the paragraph one line longer than its natural length and stores it for later. It does this in the background, *without* interfering with how TEX breaks paragraphs into their natural length.

After TEX has broken its paragraph into its natural length, lua-widow-control appears again. Before the broken paragraph is added to the main vertical list, lua-widow-control "tags" the first and last nodes of the paragraph using a LuaTEX attribute. These attributes associate the previously-saved lengthened paragraph with the naturally-typeset paragraph on the page.

13.3. Page breaking

lua-widow-control intercepts `\box255` (the `\vbox` output by TEX) immediately before the output routine runs, after all the paragraphs have been typeset.

First, lua-widow-control looks at the `\outputpenalty` of the page or column. If the page was broken at a widow or orphan, the `\outputpenalty` will be equal to either the `\widowpenalty` or the `\clubpenalty`. If the `\outputpenalty` does not indicate a widow or orphan, lua-widow-control will stop and return `\box255` unmodified to the output routine, and TEX continues as normal.

Otherwise, we assume that we have a widow or orphan on the page, meaning that we should lengthen the page by 1 line. We iterate through the list of saved paragraphs to find the lengthened paragraph with the least cost. After we've selected a good paragraph, we traverse through the page to find the original version of this paragraph—the one that unmodified TEX originally typeset. Having found the original paragraph, we splice in the lengthened paragraph in place of the original.

Since the page is now 1 line longer than it was before, we pull the last line off the page to bring it back to its original length, and place that line onto the top of TEX's "recent contributions" list. When the next page begins, this line will be inserted before all other paragraphs, right at the top. Now, we can return the new, widow-free page (updated `\box255`) to the output routine, which proceeds as normal.

13.4. Footnotes

Earlier versions of lua-widow-control completely ignored inserts. This meant that if a moved line had associated footnotes, lua-widow-control would move the "footnote mark" but not the associated "footnote text". lua-widow-control now handles footnotes correctly through the mechanism detailed in the next section.

13.4.1. Inserts

Before we go into the details of how lua-widow-control handles footnotes, we need to look at what footnotes actually are to TEX. Every `\footnote` command ultimately expands to something like `\insert`⟨*class*⟩`{`⟨*content*⟩`}`, where ⟨*class*⟩ is an insertion class number, defined as `\footins` in this case (in Plain TEX and LATEX). Inserts can be found in horizontal mode (footnotes) or in vertical mode (`\topins` in Plain TEX and floats in LATEX), but they cannot be inside boxes. Each of these insert types is assigned a different class number, but the mechanism is otherwise identical. lua-widow-control treats all inserts identically, although it safely ignores vertical mode inserts since they are only ever found between paragraphs.

But what does `\insert` do exactly? When TEX sees an `\insert` primitive in horizontal mode (when typesetting a paragraph), it does two things: first, it processes the insert's content and saves it invisibly just below the current line. Second, it effectively adds the insert content's height to the height of the material on the current page. Also, for the first insert on a page, the glue in `\skip`⟨*class*⟩ is added to the current height. All this is done to ensure that there is sufficient room for the insert on the page whenever the line is output onto the page.

If there is absolutely no way to make the insert fit on the page—say, if you placed an entire paragraph in a footnote on the last line of a page—then TEX will begrudgingly "split" the insert, placing the first part on the current page and "holding over" the second part until the next page.

There are some other TEXnicalities involving `\count`⟨*class*⟩ and `\dimen`⟨*class*⟩, but they mostly don't affect lua-widow-control. See Chapter 15 in *The TEXbook* or another reference for all the details.

After TEX has chosen the breakpoints for a paragraph, it adds the chosen lines one by one to the current page. Whenever the accumulated page height is "close enough" to the target page height (normally `\vsize`) the `\output` token list (often called the "output routine") is expanded.

But before `\output` is called, TEX goes through the page contents and moves the contents of any saved inserts into `\vbox`es corresponding to the inserts' classes, namely `\box`⟨*class*⟩, so `\output` can work with them.

And that's pretty much it on the engine side. Actually placing the inserts on the page is reserved for the output routine, which is defined by the format. This

too is a complicated process, although thankfully not one that lua-widow-control needs to worry about.

13.4.2. LuaMetaTEX

The LuaMetaTEX engine treats inserts slightly differently than traditional TEX engines. The first major difference is that insertions have dedicated registers; so instead of `\box`⟨*class*⟩, LuaMetaTEX has `\insertbox`⟨*class*⟩; instead of `\count`⟨*class*⟩, LuaMetaTEX has `\insertmultiplier`⟨*class*⟩; etc. The second major difference is that LuaMetaTEX will pick up inserts that are inside of boxes, meaning that placing footnotes in things like tables or frames should mostly just work as expected.

There are also a few new parameters and other minor changes, but the overall mechanism is still quite similar to traditional TEX.

13.4.3. Paragraph breaking

As stated in the original article (Chernoff, 2022), lua-widow-control intercepts TEX's output immediately before the output routine. However, this is *after* all the inserts on the page have been processed and boxed. This is a bit of a problem because if we move a line to the next page, we need to move the associated insert; however, the insert is already gone.

To solve this problem, immediately after TEX has naturally broken a paragraph, lua-widow-control copies and stores all its inserts. Then, lua-widow-control tags the first element of each line (usually a glyph) with a LuaTEX attribute that contains the indices for the first and last associated insert. lua-widow-control also tags each line inside the insert's content with its corresponding index so that it can be found later.

13.4.4. Page breaking

Here, we follow the same algorithm as in the original article (Chernoff, 2022). However, when we move the last line of the page to the next page, we first need to inspect the line to see if any of its contents have been marked with an insert index. If so, we need to move the corresponding insert to the next page. To do so, we unpack the attributes value to get all the inserts associated with this line.

Using the stored insert indices and class, we can iterate through `\box`⟨*class*⟩ and delete any lines that match one of the current line's indices. We also need to iterate through the internal TEX box `hold_head`—the box that holds any inserts split onto the next page—and delete any matching lines. We can safely delete any of these lines since they are still stored in the original `\insert` nodes that we copied earlier.

Now, we can retrieve all of our previously-stored inserts and add them to the next page, immediately after the moved line. Then, when TEX builds that page, it will find these inserts and move their contents to the appropriate boxes

14. Choosing the "best" paragraph

As we discussed previously, lua-widow-control lengthens the paragraph with the lowest cost. However, assigning a cost to each paragraph is not quite as simple as it sounds. Before we look at how lua-widow-control assigns costs, let's look at how TEX scores paragraphs when breaking them naturally.

14.1. How TEX scores paragraphs

All glue in TEX has a certain natural size: the size that it would be in an ideal scenario. However, most glue also has stretch and shrink components so that the glue can change in size to adapt to its surroundings. For each line, TEX individually sums the total stretch/shrink for the line and the stretch/shrink that was actually used. We define the stretch/shrink ratio r as the quotient of the stretch/shrink used and the stretch/shrink available. Then the badness b of a line is approximately defined as

$$b = 100r^3.$$

This is the badness referenced in the commonly-seen `Underfull \hbox (badness 1234)` warnings that TEX produces.

TEX calculates the badness for each line individually; however, we also need to assess the paragraph as a whole. To do so, TEX defines the demerits for a whole paragraph d as approximately[5] the sum of the squared badnesses for each line. The natural paragraph that TEX breaks is the one that minimizes d.

One important thing to realize is that demerits grow incredibly fast: demerits are proportional to the *sixth* power of glue stretch. This means that you can expect to see extremely large demerit values, even for a relatively "good" paragraph.

14.2. Possible cost functions

Now, let's return to how lua-widow-control assigns costs to each paragraph. This is surprisingly more complicated than it sounds, so we'll go through a few possible cost functions first.

Here, we use c for the cost of a paragraph, d for the total demerits, and l for the number of lines (`\prevgraf`).

[5]We ignore any additional demerits or penalties that TEX may add.

14.2.1. The original implementation
The original implementation of lua-widow-control used the simple cost function

$$c = d.$$

This cost function works reasonably well, but has one major issue: it doesn't take into account the number of lines in the paragraph. The demerits for a paragraph is the sum of the demerits for each line. This means this cost function will prefer using shorter paragraphs since they tend to have fewer demerits. However, long paragraphs tend to have much more available glue stretch, so this strategy can lead to suboptimal solutions.

14.2.2. Scaling by the number of lines
Once I realized this issue, I tried correcting it by dividing by the number of lines in the paragraph to get the average demerits instead of the total demerits:

$$c = \frac{d}{l}$$

This works better than the previous function, but still has an issue. If we have a fairly bad ten-line paragraph with total demerits $10d$ and an almost-equally bad two-line paragraph with total demerits $2d + 1$, then by this cost function, the ten-line paragraph will have a lower cost and will be chosen. This means that our page now has ten bad lines instead of two bad lines, which is not ideal.

14.2.3. Current implementation
Our first cost function, $c = dl^0$, doesn't consider the number of lines at all, while our second cost function, $c = dl^{-1}$, considers the number of lines too much. Splitting the difference between the two functions, we get the current implementation:

$$c = \frac{d}{\sqrt{l}}$$

This solves the issue with the previous function, but it adds a new issue: given a short paragraph with a large number of demerits per line and a long paragraph with fairly few average demerits per line, this function will often choose the shorter line. Although this sounds bad, in practice it gives much better results since very bad short paragraphs are *much* less noticeable than slightly bad long paragraphs.

Of course, this new function may still not be quite perfect. lua-widow-control uses the `lwc.paragraph_cost(demerits, lines)` Lua function to calculate a paragraph's cost; if you want, you can redefine this function to anything that you want.

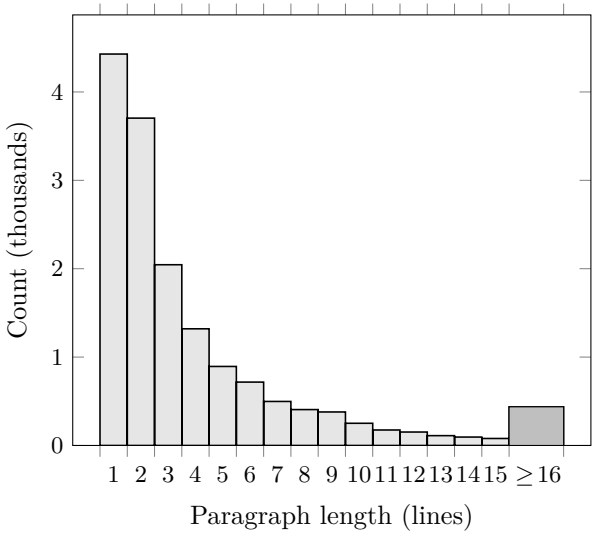

Figure 4: Histogram of natural paragraph lengths in the sample text.

15. Quantitative analysis

Let's look at some statistics for lua-widow-control. For testing, I downloaded the top ten books on *Project Gutenberg*,[6] converted them to LaTeX using pandoc, concatenated them into a single article file, and compiled twice. This gives us a PDF with 1 381 pages, 15 692 paragraphs, 61 865 lines, and 399 widows and orphans (if they aren't removed).

This is a fairly challenging test: almost every third page has a widow or orphan, over half of the paragraphs have two lines or fewer, and the text block is set to the fairly wide article defaults. An average document is much less challenging for lua-widow-control, so we can consider this to be a worst-case scenario.

15.1. Widows and orphans removed

When we run LaTeX with its default settings on the file, 179 (47%) of the widows and orphans are removed. When we add lua-widow-control with default settings, we remove 392 (98%). Switching to strict mode, we can only remove 52 (13%) of the widows and orphans. In balanced mode, we remove 348 (87%). See Figure 5 for a visual comparison.

[6] *Frankenstein*, *Pride and Prejudice*, *Alice's Adventures in Wonderland*, *The Great Gatsby*, *The Adventures of Sherlock Holmes*, *Simple Sabotage Field Manual*, *A Tale of Two Cities*, *The Picture of Dorian Gray*, *Moby Dick*, and *A Doll's House*.

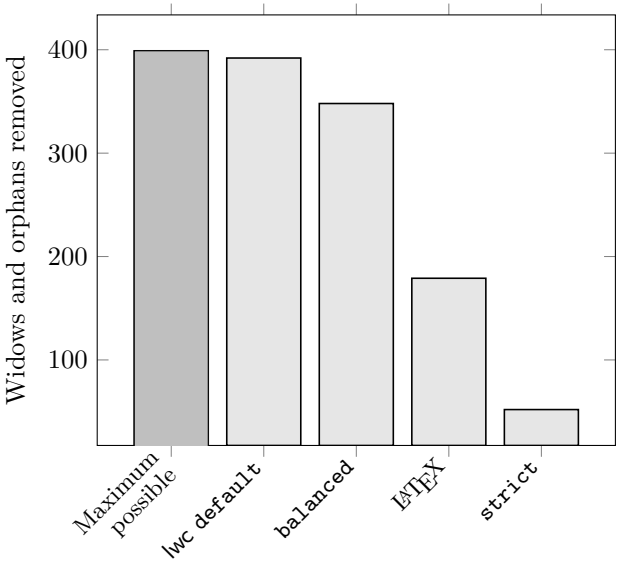

Figure 5: The number of widows and orphans removed by each method.

15.2. Paragraph costs

The last section showed us that lua-widow-control is quite effective at removing widows and orphans, so now let's look at the paragraphs that lua-widow-control expands. As TEX processes a document, lua-widow-control is recording the costs for the naturally-broken and expanded versions of each paragraph in the document. Costs don't mean that much on their own, but a lower cost is always better.

As you can see in Figure 6, the lengthened paragraphs tend to have *much* higher costs than the naturally-broken paragraphs. This is not surprising, since (as we've seen) a paragraph's demerits scale with the sixth power of glue stretch, so even a small amount of glue stretch can cause a huge increase in demerits.

The empty space on the left of the "long" line is from the paragraphs that lua-widow-control was unable to lengthen at any cost. LuaTEX assigns these paragraphs zero demerits, so they disappear on a logarithmic plot.

15.3. Lengthening vs. shortening paragraphs

Figure 7 shows the number of paragraphs that lua-widow-control could potentially stretch or shrink. The one-line paragraphs are broken out separately since this test sample has an anomalous number of them. Otherwise, we can see that lua-widow-control is capable of stretching the majority of paragraphs.

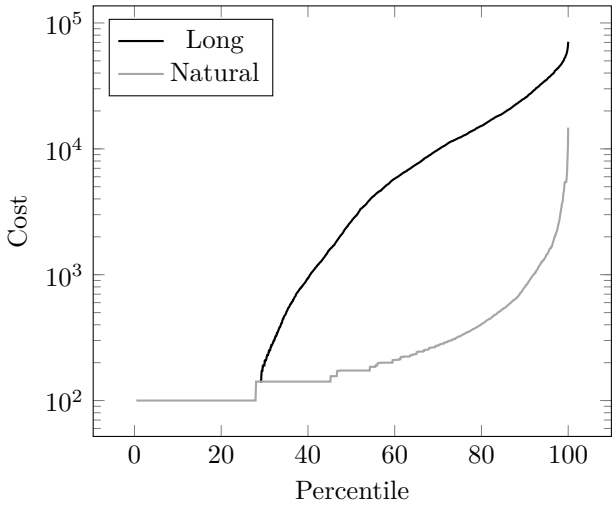

Figure 6: Paragraph costs by percentile rank for naturally-broken and one-line lengthened paragraphs.

We can also see that of non-single-line paragraphs, only about 8% of paragraphs can only be shrunk (the last segment of Figure 7), and this is in a document where 13% of paragraphs have at least eight lines. Most documents rarely have such long paragraphs, and it is these long paragraphs that are the easiest to shrink.

Because of this, lua-widow-control doesn't even attempt to shrink paragraphs; it only stretches them.

16. Known issues

lua-widow-control is quite stable these days. At this point, all *known* bugs have been resolved; some bugs certainly still remain, but I'd feel quite confident using lua-widow-control in your everyday documents. There are, however, some fundamental limitations due to how lua-widow-control operates:

- When a three-line paragraph is at the end of a page forming a widow, lua-widow-control will remove the widow; however, it will leave an orphan. This issue is inherent to any process that removes widows through paragraph expansion and is thus unavoidable. Orphans are considered to be better than widows (Bringhurst, 2004), so this is still an improvement.
- Sometimes a widow or orphan cannot be eliminated because no paragraph has enough stretch. Sometimes this can be remediated by increasing lua-widow-control's `\emergencystretch`; however, some pages just don't have any suitable paragraph.

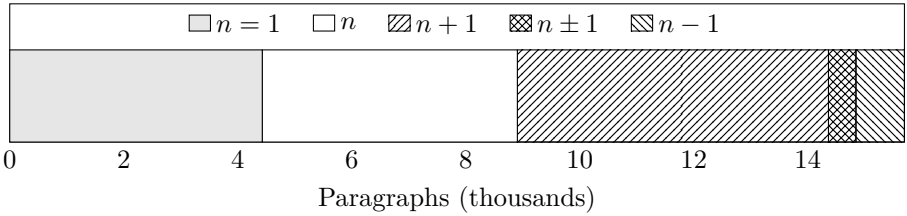

Figure 7: The number of paragraphs in the test sample that (respectively) have exactly one line, cannot be stretched or shrunk, can be only stretched by one line, can be either stretched or shrunk, and can be only shrunk.

Long paragraphs with short words tend to be stretchier than short paragraphs with long words since these long paragraphs have more interword glue. Narrow columns also stretch more easily than wide columns since you need to expand a paragraph by less to make a new line.

- lua-widow-control only attempts to expand paragraphs on a page with a widow or orphan. A global system like in "A general framework for globally optimized pagination" (Mittelbach, 2018a) would solve this; however, this is both NP-complete (Plass, 1981) and impossible to solve in a single pass. Very rarely would such a system remove widows or orphans that lua-widow-control cannot.
- lua-widow-control won't properly move footnotes if there are multiple different "classes" of inserts on the same line. To the best of my knowledge, this shouldn't happen in any real-world documents. If this happens to be an issue for you, please let me know; this problem is relatively easy to fix, although it will add considerable complexity for what I think isn't a real issue.

17. Conclusion

All this probably makes lua-widow-control look quite complicated, and this is true to some extent. However, this complexity is hidden from the end user: as stated at the outset, most users merely need to place `\usepackage{lua-widow-control}` in their LaTeX document preamble, and lua-widow-control will remove all the troublesome widows and orphans, without needing any manual intervention.

Should you have any issues, questions, or suggestions for lua-widow-control, please visit the project's GitHub page: `https://github.com/gucci-on-fleek--lua-widow-control`. Any feedback is greatly appreciated!

References

AMBROSE, G.; HARRIS, P., 2007. *The Layout Book*. Bloomsbury Academic. Advanced Level Series. ISBN 9782940373536.

BRINGHURST, R., 2004. *The Elements of Typographic Style*. 3rd. Hartley & Marks.

BROWN, Karl, 1948a. The Typographical Widow: Who is she? What is she? *Bulletin of the New York Public Library*. Vol. 52, no. 1, pp. 3–25. Available from: `https://hdl.handle.net/2027/uc1.b3310084`.

BROWN, Karl, 1948b. The Typographical Widow: Encore: Encore. *Bulletin of the New York Public Library*. Vol. 52, no. 9, pp. 458–466. Available from: `https://hdl.handle.net/2027/uc1.b3310084`.

CHERNOFF, Max, 2022. Automatically removing widows and orphans with `lua-widow-control`. *TUGboat*. Vol. 43, no. 1, pp. 28–39. Available from DOI: `10.47397/tb/43-1/tb133chernoff-widows`.

HUNT, R., 2020. *Advanced Typography: From Knowledge to Mastery*. Bloomsbury Publishing. ISBN 9781350055926.

ISAMBERT, Paul, 2010. Strategies against widows. *TUGboat*. Vol. 31, no. 1, pp. 12–17. ISSN 0896-3207. Available from: `https://tug.org/TUGboat/tb31-1/tb97isambert.pdf`.

JEREMIE, 2017. *Paragraph callback to help with widows/orphans hand tuning* [online]. 2017-07. [visited on 2022-11-08]. Available from: `https://tex.stackexchange.com/q/372062`.

KNUTH, Donald E., 1989. The new versions of TEX and METAFONT. *TUGboat*. Vol. 10, no. 3, pp. 325–328. ISSN 0896-3207. Available from: `https://tug.org/TUGboat/tb10-3/tb25knut.pdf`.

KNUTH, Donald E., 2021. *The TEXbook*. Addison–Wesley.

KRÜGER, Marcel, 2022. *luametalatex* [online]. 2022-10. [visited on 2022-11-08]. Available from: `https://github.com/zauguin/luametalatex`.

MITTELBACH, Frank, 2018a. A general framework for globally optimized pagination. *Computational Intelligence*. Vol. 35, no. 2, pp. 242–284. Available from: `https://doi.org/10.1111/coin.12165`.

MITTELBACH, Frank, 2018b. Managing forlorn paragraph lines (a.k.a. widows and orphans) in LATEX. *TUGboat*. Vol. 39, no. 3, pp. 246–251. ISSN 0896-3207. Available from: `https://tug.org/TUGboat/tb39-3/tb123mitt-widows.pdf`.

MITTELBACH, Frank, 2021. *The widows-and-orphans package* [online]. 2021-03. [visited on 2022-11-08]. Available from: `https://ctan.org/pkg/widows-and-orphans`.

MOXON, Joseph, 1683. *Mechanick exercises: The doctrine of handy-works applied to the art of printing*. Vol. 2. London. Available from: `https://archive.org/details/mechanickexercis00moxo_0`.

OLŠÁK, Petr, 1997. *TEXbook naruby. [TEXbook inside out]*. Brno, Czech Republic: Konvoj. ISBN 80-85615-64-9. Available from: `https://petr.olsak.net/ftp/olsak/tbn/tbn.pdf`.

OXFORD ENGLISH DICTIONARY, 2021a. *club, n.* [online]. Oxford University Press, 2021-09 [visited on 2022-11-08]. Available from: `https://www.oed.com/view/Entry/34788`.

OXFORD ENGLISH DICTIONARY, 2021b. *line at end of paragraph* [online]. Oxford University Press, 2021-12 [visited on 2022-11-08]. Available from: https://www.oed.com/view/th/class/195380.

OXFORD ENGLISH DICTIONARY, 2021c. *widow, n.* [online]. Oxford University Press, 2021-12 [visited on 2022-11-08]. Available from: https://www.oed.com/view/Entry/228912.

PLASS, Michael Frederick, 1981. *Optimal pagination techniques for automatic typesetting systems*. Available from: https://tug.org/docs/plass/plass-thesis.pdf. PhD thesis. Stanford University.

SALTZ, I., 2019. *Typography Essentials Revised and Updated: 100 Design Principles for Working with Type*. Rockport Publishers. ISBN 9781631596483.

THÀNH, Hàn Thế, 2001. *Micro-typographic extensions to the TEX typesetting system*. Brno. Available from: http://www.pragma-ade.nl/pdftex/thesis.pdf. PhD thesis. The Faculty of Informatics, Masaryk University.

THE $\mathcal{NTS}$ TEAM, 1998. *The ε-TEX manual* [online]. [visited on 2022-11-08]. Available from: https://ctan.org/pkg/etex.

WERMUTH, Udo, 2018. Experiments with \parfillskip. *TUGboat*. Vol. 39, no. 3, pp. 276–303. ISSN 0896-3207. Available from: https://tug.org/TUGboat/tb39-3/tb123wermuth-parfillskip.pdf.

Automatické odstraňování vdov a sirotků pomocí balíčku lua-widow-control

Balíček lua-widow-control pro LuaTEX/LuaLATEX/ConTEXt/OpTEX odstraňuje vdovy a sirotky bez dalšího zásahu uživatele. Využívá přitom sílu LuaTEXu a přitom nenatahuje žádné vertikální mezery a ani nezkracuje stránky nebo sloupce. Namísto toho balíček automaticky prodlužuje některý z odstavců na té stránce nebo sloupci, kde by se vdova nebo sirotek vyskytli.

Pro použití balíčku postačí většině uživatelů LATEXu uvést v preambuli dokumentu \usepackage{lua-widow-control}. Žádné další změny v dokumentu nejsou zapotřebí.

Klíčová slova: LuaTEX, vdova, sirotek

Max Chernoff, mseven at telus dot net