

Automatically removing widows and orphans with lua-widow-control

Max Chernoff

Abstract

The lua-widow-control package, for plain Lua \TeX /Lua \LaTeX /Con \TeX t/Op \TeX , removes widows and orphans without any user intervention. Using the power of Lua \TeX , it does so *without* stretching any 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 lua-widow-control, all that most users need do is place `\usepackage{lua-widow-control}` in their preamble. No further changes are required.

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 and 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 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.

Widow	Orphan
A widow occurs when the last line of a paragraph is placed on a page separate from where it begins.	An orphan is when the first line of a paragraph occurs on the page before all of 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.

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 [13], a printers manual from 1683, we have:

Nor do good *Compofiters* account it good Workmanfhip to begin a *Page* with a *Break-line*, unlefs it be a very fhort *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)

3.1 Widows

However, the terms “widow” and “orphan” are much newer. 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 [3] —

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. [3]

Contrast this with the modern definition:

Typography. A short line of text (usually one consisting of one word or part of a word) which falls undesirably at the end of a paragraph, esp. one set at the top of a page or column. [16]

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” [3], 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 [2, 3, 4, 6, 9, 12, 14, 16]; 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! [1, 3, 5, 14, 18] This usage is plain wrong; nevertheless, it is sufficiently common that you need to be careful when you see the terms “widow” and “orphan”.

3.3 Clubs

The T_EXbook 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 T_EXbook* — 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”. [3, 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.’ [3, 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. [15]

I spent a few hours searching through Google Books and my university library catalogue, but I could not find a single additional source. If anyone has any more information on the definition of a “club line” or why Knuth chose to use this archaic Scottish term in T_EX, please let me know!

4 Pagination in T_EX

Let’s move on to looking at how T_EX 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 T_EX breaks pages and columns. For a full description, you should consult Chapter 15 of *The T_EXbook* [9] (“How T_EX Makes Lines into Pages”); however, this goes into much more detail than most users require, so here is a *very* simplified summary of T_EX’s page breaking algorithm:

T_EX fills the page with lines and other objects until the next object will no longer fit. Once no more objects will fit, T_EX will align the bottom of the last line with the bottom of the page by stretching any available vertical spaces if (in L^AT_EX) `\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, L^AT_EX 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, T_EX 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 T_EX 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 T_EX deals with widows and orphans.

Due to the penalties attached to widows and orphans, T_EX tries to avoid them. Widows and orphans with small penalties attached — like L^AT_EX’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 [9] (“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 able to automatically remove widows and orphans without stretching any glue or shortening any pages.

The articles “Strategies against widows” by Paul Isambert [6] and “Managing forlorn paragraph lines” by Frank Mittelbach [11] 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 . . .” by jeremie [7] contains a file `widow-assist.lua` that automatically detects which paragraphs can be safely shortened or lengthened by one line. Mittelbach’s `widows-and-orphans` package [12] 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 by Mittelbach [10] suggests an fully-automated solution to remove widows and orphans. This would seem to offer a complete solution; however, it requires multiple passes, an external tool, and has not yet been publicly released.

Ignore

<p><code>lua-widow-control</code> can remove most widows and orphans from a document, <i>without</i> stretching any glue or shortening any pages.</p> <p>It does so by automatically lengthening a paragraph on a page where a widow or orphan would otherwise occur. While \TeX breaks paragraphs into their natural length, <code>lua-widow-control</code> is breaking the paragraph 1 line longer than its natural length. \TeX's paragraph is output to the page, but <code>lua-widow-control</code>'s paragraph is just stored for later. When a widow or orphan occurs, <code>lua-widow-control</code> can take over. It selects the previously-saved paragraph with the least badness; then, it replaces \TeX's paragraph with its saved paragraph. This increases the text block height of the page by 1 line.</p> <p>Now, the last line of the current page can be pushed to the top of the next page. This removes the widow or the orphan without creating any additional work.</p>

```
\parskip=0pt
\clubpenalty=0
\widowpenalty=0
```

Stretch

<p><code>lua-widow-control</code> can remove most widows and orphans from a document, <i>without</i> stretching any glue or shortening any pages.</p> <p>It does so by automatically lengthening a paragraph on a page where a widow or orphan would otherwise occur. While \TeX breaks paragraphs into their natural length, <code>lua-widow-control</code> is breaking the paragraph 1 line longer than its natural length. \TeX's paragraph is output to the page, but <code>lua-widow-control</code>'s paragraph is just stored for later. When a widow or orphan occurs, <code>lua-widow-control</code> can take over. It selects the previously-saved paragraph with the least badness; then, it replaces \TeX's paragraph with its saved paragraph. This increases the text block height of the page by 1 line.</p> <p>Now, the last line of the current page can be pushed to the top of the next page.</p> <p>This removes the widow or the orphan without creating any additional work.</p>
--

```
\parskip=0pt plus 1fill
\clubpenalty=10000
\widowpenalty=10000
```

Shorten

<p><code>lua-widow-control</code> can remove most widows and orphans from a document, <i>without</i> stretching any glue or shortening any pages.</p> <p>It does so by automatically lengthening a paragraph on a page where a widow or orphan would otherwise occur. While \TeX breaks paragraphs into their natural length, <code>lua-widow-control</code> is breaking the paragraph 1 line longer than its natural length. \TeX's paragraph is output to the page, but <code>lua-widow-control</code>'s paragraph is just stored for later. When a widow or orphan occurs, <code>lua-widow-control</code> can take over. It selects the previously-saved paragraph with the least badness; then, it replaces \TeX's paragraph with its saved paragraph. This increases the text block height of the page by 1 line.</p> <p>Now, the last line of the current page can be pushed to the top of the next page.</p> <p>This removes the widow or the orphan without creating any additional work.</p>
--

```
\parskip=0pt
\clubpenalty=10000
\widowpenalty=10000
```

lua-widow-control

<p><code>lua-widow-control</code> can remove most widows and orphans from a document, <i>without</i> stretching any glue or shortening any pages.</p> <p>It does so by automatically lengthening a paragraph on a page where a widow or orphan would otherwise occur. While \TeX breaks paragraphs into their natural length, <code>lua-widow-control</code> is breaking the paragraph 1 line longer than its natural length. \TeX's paragraph is output to the page, but <code>lua-widow-control</code>'s paragraph is just stored for later. When a widow or orphan occurs, <code>lua-widow-control</code> can take over. It selects the previously-saved paragraph with the least badness; then, it replaces \TeX's paragraph with its saved paragraph. This increases the text block height of the page by 1 line.</p> <p>Now, the last line of the current page can be pushed to the top of the next page.</p> <p>This removes the widow or the orphan without creating any additional work.</p>
--

```
\usepackage{lua-widow-control}
```

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

`lua-widow-control` is essentially a combination of `widow-assist.lua` [7] and `widows-and-orphans` [12] (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 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 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 $\text{MiK}\TeX$ and $\text{T}\TeX$ Live, although you will need recent versions of either.

You may also download `lua-widow-control` manually from either CTAN,¹ the Con $\text{T}\TeX$ t Garden,² or GitHub,³ although it is best if you can install it through your $\text{T}\TeX$ distribution.

As its name may suggest, `lua-widow-control` requires $\text{Lua}\TeX$ ⁴ regardless of the format used. With that in mind, using `lua-widow-control` is quite simple:

Plain $\text{T}\TeX$	<code>\input lua-widow-control</code>
Op $\text{T}\TeX$	<code>\load[lua-widow-control]</code>
$\text{L}\text{A}\text{T}\text{E}\text{X}$	<code>\usepackage{lua-widow-control}</code>
Con $\text{T}\text{E}\text{X}\text{t}$	<code>\usemodule[lua-widow-control]</code>

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 $\text{T}\TeX$ /Op $\text{T}\TeX$ Some options are set by modifying a register, while others must be set manually using `\directlua`.

$\text{L}\text{A}\text{T}\text{E}\text{X}$ Options can be set either as package options or at any point in the document with `\lwcsetup`.

Con $\text{T}\text{E}\text{X}\text{t}$ Always use `\setuplwc`.

¹ ctan.org/pkg/lua-widow-control

² [modules.contextgarden.net/cgi-bin/module.cgi/action=view/id=127](https://modules.contextgarden.net/cgi-bin/module.cgi?action=view/id=127)

³ github.com/gucci-on-fleek/lua-widow-control/releases/latest/

⁴ Or $\text{LuaMeta}\text{T}\text{E}\text{X}$ in the case of Con $\text{T}\text{E}\text{X}\text{t}$.

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 lengthen 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 [8], 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.

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 naively 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      \directlua{lwc.
                    nobreak_behaviour="<value>"}
LATeX              \lwcsetup{nobreak=<value>}
ConTeXt              \setuplwc[nobreak=<value>]
```

keep	split	warn
	Heading	Heading
Heading The first line text text text	The first line text text text last line.	The first line text text text last line.

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

The default value, `keep`, *keeps* 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 `\nobreaks`; the disadvantage is that moving the section heading can create a large blank space at the end of the page.

The value `split` *splits* up the section heading and the orphan by moving the orphan to the next page while 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.

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.

Currently, these presets are L^AT_EX-only.

```
LATEX \lwcsetup{<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 T_EX 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 T_EX footprint. It doesn’t modify the output routine, inserts/floats,

Table 1: 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

`\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 matter for Plain \TeX or \LaTeX , where even most Lua-based packages don't depend on the node list structure; nevertheless, there are a few issues with Con \TeX t.

Simple Con \TeX t documents tend to be fine, but many advanced Con \TeX t features rely heavily on Lua and can thus be disturbed by lua-widow-control. This is not a huge issue — the lua-widow-control manual is written in Con \TeX t — but lua-widow-control is inevitably more reliable with Plain \TeX and \LaTeX than with Con \TeX t.

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 Lua \TeX -based formats: Plain Lua \TeX , Lua \LaTeX , Con \TeX t MkIV, Con \TeX t MkXL/LMTX, and Op \TeX . Unless otherwise documented, all features should work equally well in all formats.

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 [9].

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.

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 [20] generalize these commands with a syntax similar to `\parshape`: with `\widowpenalties` you can set the penalty between the last, second last, and n th last lines of a paragraph; `\interlinepenalties` and `\clubpenalties` behave similarly.

lua-widow-control 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 [9]. lua-widow-control does this automatically, but instead of using ties or `\hboxes`, it uses the `\parfillskip` parameter [9, 21]. 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 `0pt plus 0.8\hspace`. 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 parameters `\clubpenalty`, `\widowpenalty`, and `\brokenpenalty` 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 Lua \TeX 's microtypographic extensions [19]. 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 Lua \TeX 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.

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 approximately⁵ 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 very 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:

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

I didn't arrive at this function through any sort of scientific testing; rather, I picked the simplest function that I could think of that satisfies the following two properties:

- Given a long paragraph and a short paragraph with different average badnesses per line, prefer the one with the least average badness.
- Given two paragraphs with equal average badnesses per line, prefer the shorter one.

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

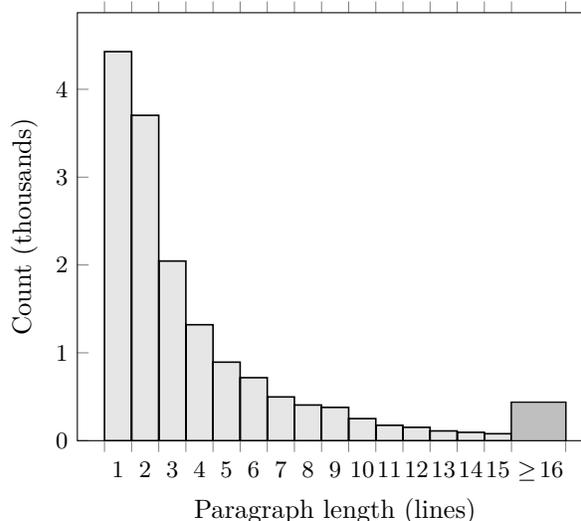


Figure 4: Histogram of natural paragraph lengths in the sample text.

file, and compiled twice. This gives us a PDF with 1381 pages, 15692 paragraphs, 61865 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.

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.

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.*

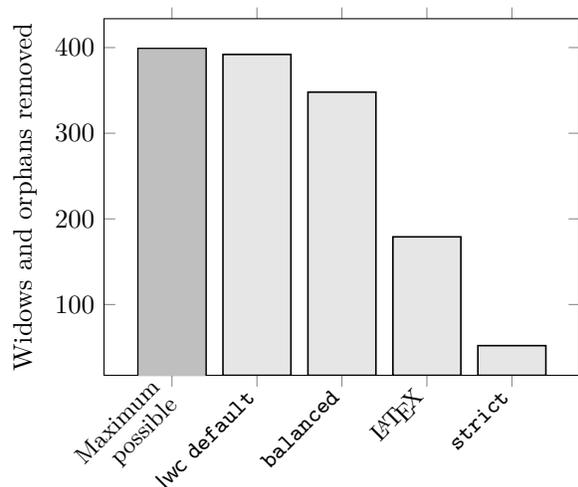


Figure 5: The number of widows and orphans removed by each method.

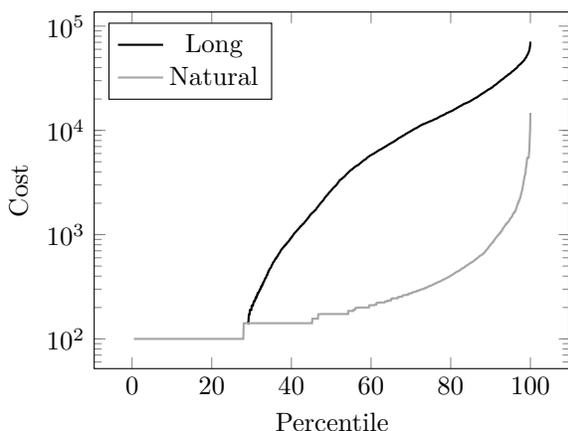


Figure 6: Paragraph costs by percentile rank for naturally-broken and one-line lengthened paragraphs.

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

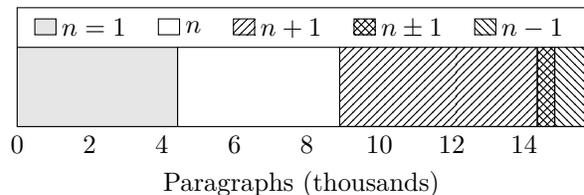


Figure 7: The number of paragraphs in the test sample that (respectively) have exactly one line, cannot be stretched or shrunk, can be only stretched by one line, can be either stretched or shrunk, and can be only shrunk.

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.

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, a few issues remain:

- 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 [2], 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.

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.

- When running under LuaMetaTeX (ConTeXt), the log may contain many lines like “`luatex warning > tex: left parfill skip is gone`”.

These messages are completely harmless (although admittedly quite annoying).

- \TeX may warn about overfull `\vboxes` on pages where `lua-widow-control` removed a widow or orphan. This happens due to the way that `lua-widow-control` corrects for the `\prevdepth` when replacing paragraphs. It does not actually produce an overfull `vbox`, but there is a warning nevertheless. You can set `\vfuuzz=2.5pt` to hide the warning.
- `lua-widow-control` only attempts to expand paragraphs on a page with a widow or orphan. A global system like in [10] would solve this; however, this is both NP-complete [17] and impossible to solve in a single pass. Very rarely would such a system remove widow or orphans that `lua-widow-control` cannot.

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: github.com/gucci-on-fleek/lua-widow-control. Any feedback is greatly appreciated!

References

- [1] G. Ambrose, P. Harris. *The Layout Book*. Advanced Level Series. Bloomsbury Academic, 2007.
- [2] R. Bringhurst. *The Elements of Typographic Style*. Hartley & Marks, 3rd ed., 2004.
- [3] K. Brown. The typographical widow. *Bulletin of the New York Public Library* 52(1):3–25, Jan. 1948. hdl.handle.net/2027/uc1.b3310084
- [4] K. Brown. The typographical widow: Encore. *Bulletin of the New York Public Library* 52(9):458–466, Sept. 1948. hdl.handle.net/2027/uc1.b3310084
- [5] R. Hunt. *Advanced Typography: From Knowledge to Mastery*. Bloomsbury Publishing, 2020.
- [6] P. Isambert. Strategies against widows. *TUGboat* 31(1):12–17, 2010. tug.org/TUGboat/tb31-1/tb97isambert.pdf
- [7] jeremie. Paragraph callback to help with widows/orphans hand tuning, August 2017. tex.stackexchange.com/q/372062
- [8] D.E. Knuth. The new versions of \TeX and METAFONT. *TUGboat* 10(3):325–328, Nov. 1989. tug.org/TUGboat/tb10-3/tb25knuth.pdf
- [9] D.E. Knuth. *The \TeX book*. Addison–Wesley, 2021.
- [10] F. Mittelbach. A general framework for globally optimized pagination. *Computational Intelligence* 35(2):242–284, Mar. 2018. doi.org/10.1111/coin.12165
- [11] F. Mittelbach. Managing forlorn paragraph lines (a.k.a. widows and orphans) in \LaTeX . *TUGboat* 39(3):246–251, 2018. tug.org/TUGboat/tb39-3/tb123mitt-widows.pdf
- [12] F. Mittelbach. The widows-and-orphans package, March 2021. ctan.org/pkg/widows-and-orphans
- [13] J. Moxon. *Mechanick exercises*, vol. 2, 1683. archive.org/details/mechanickexercis00moxo_0
- [14] Oxford English Dictionary. line at end of paragraph. www.oed.com/view/th/class/195380
- [15] Oxford English Dictionary. club, n., Sept. 2021. www.oed.com/view/Entry/34788
- [16] Oxford English Dictionary. widow, n., Dec. 2021. www.oed.com/view/Entry/228912
- [17] M.F. Plass. *Optimal pagination techniques for automatic typesetting systems*. Ph.D. thesis, Stanford University, 1981. tug.org/docs/plass/plass-thesis.pdf
- [18] I. Saltz. *Typography Essentials Revised and Updated*. Rockport Publishers, 2019.
- [19] Hàn Thế Thành. Micro-typographic extensions to the \TeX typesetting system. *TUGboat* 21(4):317–317, Dec. 2000. tug.org/TUGboat/tb21-4/tb69thanh.pdf
- [20] The $\mathcal{N}\mathcal{T}\mathcal{S}$ Team. The ε - \TeX manual, Feb. 1998. ctan.org/pkg/etex
- [21] U. Wermuth. Experiments with `\parfillskip`. *TUGboat* 39(3):276–303, 2018. tug.org/TUGboat/tb39-3/tb123wermuth-parfillskip.pdf

◇ Max Chernoff
Calgary, Alberta
Canada

Updates to “Automatically removing widows and orphans with lua-widow-control”, TUGboat 43:1

Max Chernoff

A request from *Zpravodaj*, the journal of the Czech/Slovak T_EX group, to republish the subject article led to these updates. The section numbers here correspond to those in the original article.

3.3 Clubs

In the original article, I discussed the origin of the typographical terms “widow”, “orphan”, and “club”. The first two terms are fairly well-known, but I had this to say regarding the third:

The T_EXbook 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 T_EXbook* — a compilation article about the definition of “widow” — that mentions a “club line” [...]

I spent a few hours searching through Google Books and my university library catalogue, but I could not find a single additional source. If anyone has any more information on the definition of a “club line” or why Knuth chose to use this archaic Scottish term in T_EX, please let me know!

Conveniently, Don Knuth — the creator of T_EX — read my plea 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 T_EX78 had only “\chpar4” to change the “widowpenalty”. Sometime between then and T_EX82 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?

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

9 Options

The overview to the “options” section stated that:

Plain T_EX/OpT_EX Some options are set by modifying a register, while others must be set

manually using `\directlua`.

However, this is no longer true. Now, commands are provided for all options in all formats, so you no longer need to use ugly `\directlua` commands in your documents. The old commands still work, although they will likely be removed at some point in the future.

9.5 Penalties

`\brokenpenalty` now also exists as a L^AT_EX and ConT_EXt key. `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

The Plain/OpT_EX command is now:

```
\lwc nobreak{<value>}
```

9.8 Draft mode

Since v2.2.0, `lua-widow-control` has a “draft mode” which shows how `lua-widow-control` processes pages.

Plain T _E X/OpT _E X	<code>\lwc draft 1</code>
L ^A T _E X	<code>\lwc setup{draft}</code>
ConT _E Xt	<code>\setuplwc [draft=start]</code>

Draft mode has been used for typesetting this article. It 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 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 pdfL^AT_EX and LuaL^AT_EX, 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

3953 acceptable to you.

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

```
Plain TEX/      \lwcshowcosts 1
OpTEX         \lwcshowcolours 0
LATEX        \lwcsetup{showcosts=true}
                \lwcsetup{showcolours=false}
ConTEXt       \setuplwc[showcosts=start]
                \setuplwc[showcolours=stop]
```

To demonstrate the new draft mode, I have tricked lua-widow-control into thinking that every column in this article ends in a widow, even when they actually don't. This means that lua-widow-control is attempting to expand paragraphs on every column. This gives terrible page breaks and often creates new widows and orphans, but it's a good demonstration of how lua-widow-control works.

1881

10 Presets

The original article stated that “presets are L^AT_EX-only”. However, lua-widow-control now supports presets with both L^AT_EX and ConT_EXt using the following commands:

infinite

```
LATEX  \lwcsetup{<preset>}
ConTEXt \setuplwc[<preset>]
```

infinite

11 Compatibility

This quote:

infinite

It doesn't modify [...], inserts/floats,

infinite

isn't strictly true since v2.1.2 since lua-widow-control now handles moving footnotes.

infinite

This statement is also no longer true:

infinite

there are a few issues with ConT_EXt [...] lua-widow-control is inevitably more reliable with Plain T_EX and L^AT_EX than with ConT_EXt.

27175

All issues with ConT_EXt — including grid snapping — have now been resolved. lua-widow-control should be equally reliable with all formats.

21928

11.1 Formats

In addition to the previously-mentioned formats/engines, lua-widow-control now has preliminary support for LuaMetaL^AT_EX and LuaMetaPlain.¹ Aside from a few minor bugs, the LuaMetaL^AT_EX and LuaMetaPlain versions work identically to their respective LuaL^AT_EX versions. With this addition, lua-widow-control now supports seven different format/engine

21928

¹ github.com/zauguin/luametalatex

combinations.

15279

11.3 Performance

Earlier versions of lua-widow-control had some memory leaks. These weren't noticeable for small documents, although it could cause slowdowns for documents larger than a few hundred pages. However, I have implemented a new testing suite to ensure that there are no memory leaks, so lua-widow-control can now easily compile documents > 10 000 pages long.

35410

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.

23050

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 T_EX. 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 T_EX and L^AT_EX). Inserts can be found in horizontal mode (footnotes) or in vertical mode (\topins in Plain T_EX and floats in L^AT_EX), 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.

15243

But what does \insert do exactly? When T_EX 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.

35655

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 T_EX will begrudgingly “split” the insert, placing the first part on the current page and “holding over” the second part until the next page.

1149

There are some other T_EXnicities involving

`\count<class>` and `\dimen<class>`, but they mostly don't affect `lua-widow-control`. See Chapter 15 in *The TeXbook* or some other 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 `\vboxes` 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, `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. 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.

16 Known issues

The following two bugs have now been fully resolved:

- When running under LuaMetaTeX, the log may contain [...] infinite
- TeX may warn about overfull `\vboxes` [...] 5759

The fundamental limitations previously listed still exist; however, these two bugs along with a few dozen others have all been fixed since the original article was published. 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. 2502

There is, however, one new issue: infinite

- `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. 9875

◊ Max Chernoff 9875
<https://ctan.org/pkg/lua-widow-control> 9875