

LUA-WIDOW-CONTROL

v 1.0.0

<https://github.com/gucci-on-fleek/lua-widow-control>

Lua-widow-control is a Plain T_EX/L^AT_EX/ConT_EXt package that removes widows and orphans without any user intervention. Using the power of LuaT_EX, it does so *without* stretching any glue or shortening any pages. Instead, lua-widow-control automatically lengthens a paragraph on a page where a widow or orphan would otherwise occur.

QUICK START

Install lua-widow-control by unzipping the **latest release** in your TEXMFLOCAL directory. Rebuild the filename database. Place `\usepackage{lua-widow-control}` in the preamble of your document.

For a more detailed explanation of the steps involved and a description of some of the finer points involved with using lua-widow-control, keep reading.

CONTENTS

Quick Start	1
Motivation	2
Widows and Orphans	2
T _E X's Pagination	3
Previous Work	4
Demonstration	5
Installation	6
Loading the Package	6
Advanced Usage	6
Known Issues	7
The Algorithm	8
Contributions	9
License	9
References	9
Implementation	10

Warning

Lua-widow-control works by modifying the internal structure of each page. In **rare** scenarios, this could delete or duplicate paragraphs or lines when a page break is triggered. All known bugs have been fixed, but I would still caution against using lua-widow-control in any document where a single missing line would be disastrous. Lua-widow-control should be sufficiently stable to use in typical documents.

MOTIVATION

T_EX 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 T_EX typically has only 2 ways of dealing with widows and orphans: it can either shorten a page by 1 line, or it can stretch out some vertical whitespace. T_EX was designed for mathematical and scientific typesetting, where a typical page has multiple section headings, tables, figures, and equations. For this style of document, T_EX's default behavior works quite well; however, for prose or any other document composed almost entirely of text, there is no vertical whitespace to stretch.

Since there were no ready-made and fully-automated solutions to remove widows and orphans from prosaic documents, I decided to create lua-widow-control.

WIDOWS AND ORPHANS

Widows Widows occur when when the majority of a paragraph is on one page, but the last line is on the following page. Widows are undesirable for both aesthetics and readability. Aesthetically, it looks quite odd for a lone line to be at the start of the page. Functionally, the separation of a paragraph and its last line disconnects the two, causing the reader to lose context for the widowed line.

Orphans Orphans are when the first line of a paragraph occurs on the page before the remainder of the paragraph. They are not nearly 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 do, so they tend to be ignored more often.

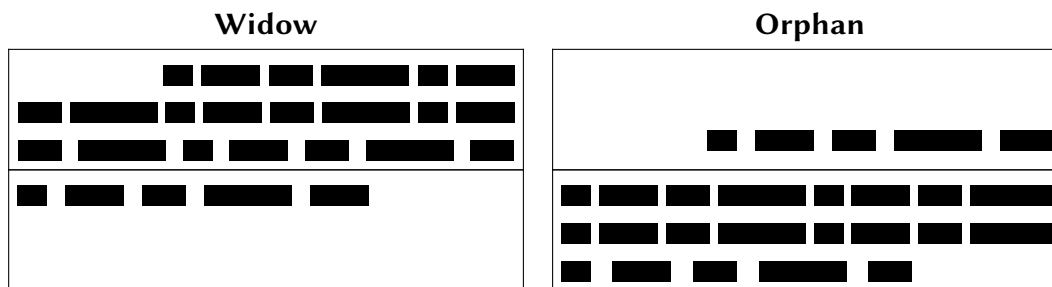


Figure 1 A visual comparison of widows and orphans.

T_EX'S PAGINATION

Algorithm It is tricky to understand how lua-widow-control works if you aren't familiar with how T_EX breaks pages. Chapter 15 of *The T_EXBook*¹ (“How T_EX Makes Lines into Pages”) is the best reference for this; however, it goes into much more detail than most users require. As a supplemental resource, I can also recommend Section 27 of *T_EXby Topic*², **available online** for free. Below follows a *very* simplified (and likely error-ridden) 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 vertical spaces.

However, some objects have penalties attached. These penalties make T_EX treat the object as if it is longer or shorter for the sake of page breaking. By default, T_EX assigns a penalties to the first and last lines of a paragraph (widows and orphans). This makes T_EX treat them as if they are larger or smaller than their actual size such that T_EX tends not to break them up.

One important note: once T_EX begins breaking a page, it only breaking a page. That is, it will never go back and modify any content that has previously been added to its *main vertical list*. That is, page breaking is a localized algorithm, without any backtracking.

Behavior Of course, this 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 treats them as if they are longer than they actually are. Widows and orphans with small penalties attached—like L^AT_EX's default values of 150—are only treated as slightly taller than 1 line, while widows and orphans with large penalties—values near 10 000—are

treated as if they are 2 lines tall. Because potential widow and orphan lines are broken as if they are taller than they actually are, T_EX will tend to group them together on the same pages.

However, when these lines are moved as a group, T_EX will have to make a page with less lines. “**Demonstration**” goes into further detail about how T_EX deals with these too-short pages. The main takeaway is that for a page exclusively filled with text, all of T_EX’s builtin solutions come with compromises.

PREVIOUS WORK

There have been a few previous attempts to improve upon T_EX’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.

*Strategies against widows*³ and *Managing forlorn paragraph lines in L^AT_EX*⁴ both begin with comprehensive discussions of the methods of preventing widows and orphans. They both agree that widows and orphans are bad and ought to be avoided; however, they each differ in solutions. *Strategies*³ proposes an output routine that reduces the length of facing pages by 1 line if necessary to remove widows and orphans while *Managing*⁴ proposes that the author manually rewrites or adjusts the `\looseness` when needed.

*Paragraph callback to help with widows/orphans hand tuning*⁵ contains a file `widow-assist.lua` that automatically detects which paragraphs can be safely shortened or lengthened by 1 line. *The widows-and-orphans package*⁶ alerts the author to the pages that contain widows or orphans. Combined, these packages make it very simple for the author to quickly remove widows and orphans by adjusting the `\looseness` values; however, it still requires the author to make manual source changes after each revision.

Lua-widow-control is essentially just a combination of `widow-assist.lua`⁵ and *widows-and-orphans*:⁶ when the `\outputpenalty` shows that a widow or orphan occurred, Lua is used to find a stretchable paragraph. What lua-widow-control adds on top of these packages is automation: lua-widow-control eliminates the requirement for any manual adjustments.

Ignore	Shorten	Stretch	Lua-widow-control
<p>Lua-widow-control 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, lua-widow-control is breaking the paragraph 1 line longer than its natural length. TeX's paragraph is output to the page, but lua-widow-control's paragraph is just stored for later. When a widow or orphan occurs, lua-widow-control can take over. It selects the previously-saved paragraph with the least badness; then, it replaces TeX's paragraph with its saved paragraph. This lengthens 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 for</p>	<p>Lua-widow-control 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, lua-widow-control is breaking the paragraph 1 line longer than its natural length. TeX's paragraph is output to the page, but lua-widow-control's paragraph is just stored for later. When a widow or orphan occurs, lua-widow-control can take over. It selects the previously-saved paragraph with the least badness; then, it replaces TeX's paragraph with its saved paragraph. This lengthens 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</p>	<p>Lua-widow-control 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, lua-widow-control is breaking the paragraph 1 line longer than its natural length. TeX's paragraph is output to the page, but lua-widow-control's paragraph is just stored for later. When a widow or orphan occurs, lua-widow-control can take over. It selects the previously-saved paragraph with the least badness; then, it replaces TeX's paragraph with its saved paragraph. This lengthens 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</p>	<p>Lua-widow-control 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, lua-widow-control is breaking the paragraph 1 line longer than its natural length. TeX's paragraph is output to the page, but lua-widow-control's paragraph is just stored for later. When a widow or orphan occurs, lua-widow-control can take over. It selects the previously-saved paragraph with the least badness; then, it replaces TeX's paragraph with its saved paragraph. This lengthens 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</p>
<p>the document's author.</p>	<p>without creating any additional work for the document's author.</p>	<p>without creating any additional work for the document's author.</p>	<p>without creating any additional work for the document's author.</p>
<p><code>\parskip=0pt</code></p> <p><code>\clubpenalty=0</code></p> <p><code>\widowpenalty=0</code></p>	<p><code>\parskip=0pt</code></p> <p><code>\clubpenalty=10000</code></p> <p><code>\widowpenalty=10000</code></p>	<p><code>\parskip=0pt</code> plus 1fill</p> <p><code>\clubpenalty=10000</code></p> <p><code>\widowpenalty=10000</code></p>	<p><code>\usepackage{lua-widow-control}</code></p>

Table 1 A visual comparison of various automated widow handling techniques.

DEMONSTRATION

Although \TeX 's page breaking algorithm is quite simple, it can lead to some fairly complex behaviors when widows and orphans are involved. The usual choices are to either ignore them, stretch some glue, or shorten the page. **Table 1** has a visual demonstration of some of these behaviors and how `lua-widow-control` differs from the defaults.

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 pretty distracting for the reader, so it is best avoided wherever possible.
Shorten	This page did not leave any widows, but it did shorten the previous page by 1 line. Sometimes this is acceptable, but usually it looks bad because each page will 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.
Stretch	<p>This page also has no widows and it has a flushed bottom margin. However, the space between each paragraph had to be stretched.</p> <p>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 behavior. However, in a page with mostly text, these paragraph gaps can look unsightly.</p> <p>In addition, this method is incompatible with typesetting on a grid since all glue stretch is quantized to the height of a line.</p>
lua-widow-control	<p>Lua-widow-control has none of these issues: it eliminates the widows in a document while keeping a flushed bottom margin and constant paragraph spacing.</p> <p>To do so, <code>lua-widow-control</code> 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 it becomes nearly imperceptible with larger widths.</p> <p>Lua-widow-control automatically finds the “best” paragraph to stretch, so the increase in interword spaces should almost always be minimal.</p>

INSTALLATION

- T_EX Live** Run `tlmgr install lua-widow-control` in a terminal, or install using the “T_EX Live Manager” GUI.
- MikT_EX** MikT_EX should automatically install the package when you include it in your document.
- Manual** Currently, ConT_EXt MKXL (luametaT_EX) users must manually install the package. Most other users will be better served by using the lua-widow-control supplied by T_EXLive and MikT_EX; however, all users may manually install the package if desired. The procedure should be fairly similar regardless of your OS, T_EX distribution, or format.
- Steps**
1. Download `lua-widow-control.tds.zip` from the **latest release** on GitHub.
 2. Unzip the release into your `TEXMFLOCAL/` directory. (You can find its location by running `kpsewhich --var-value TEXMFHOME` in a terminal)
 3. Refresh the filename database:
 - ConT_EXt: `mtxrun --generate`
 - T_EXLive: `mktexlsr`
 - MikT_EX: `initexmf --update-fndb`

LOADING THE PACKAGE

- Plain T_EX** `\input lua-widow-control`
- L^AT_EX** `\usepackage{lua-widow-control}`
- ConT_EXt** `\usemodule[lua-widow-control]`

ADVANCED USAGE

Lua-widow-control is automatically enabled with the default settings as soon as you load it. Most users should not need to configure lua-widow-control; however, the packages does provide a few commands.

- Plain T_EX**
- | | |
|--------------------------------|---|
| Enable (default) | <code>\lwcenable</code> |
| Disable | <code>\lwcdisable</code> |
| <code>\emergencystretch</code> | <code>\lwcemergencystretch = 3em</code> |

L ^A T _E X	Enable (default)	<code>\lwcenable</code>
	Disable	<code>\lwcdisable</code>
	<code>\emergencystretch</code>	<code>\setlength {\lwcemergencystretch }{3em}</code>
ConT _E Xt	Enable (default)	<code>\setuplwc [state = start]</code>
	Disable	<code>\setuplwc [state = stop]</code>
	<code>\emergencystretch</code>	<code>\setuplwc [emergencystretch = 3em]</code>
<code>\emergency stretch</code>	<p>You can configure the <code>\emergencystretch</code> used when stretching a paragraph. The default value is 3 em.</p> <p>Lua-widow-control will only use use the <code>\emergencystretch</code> when it cannot lengthen a paragraph in any other way, so it is fairly safe to set this to a large value. T_EX still accumulates badness when <code>\emergencystretch</code> is used, so its pretty rare that a paragraph that requires any <code>\emergencystretch</code> will actually be used on the page.</p>	

KNOWN ISSUES

- Lua-widow-control will rarely fail to correctly move the last line on an expanded page to the next page in documents with *very* small paper sizes. This tends to only occur with paper narrower than ≈ 8 cm or shorter than ≈ 10 cm and should not affect any typical documents.
- In multicolumn layouts, lua-widow-control will remove widows and orphans that occur between pages, but it will not remove widows and orphans between columns. To do so would likely require modifying the output routine which would be a very difficult task.
- When a 3-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 better than widows, so this is still an improvement.
- Sometimes a widow or orphan just cannot be eliminated because no paragraph has enough “stretch”. This can *sometimes* be remediated by increasing lua-widow-control's `\emergencystretch`; however, some pages just don't have enough “stretchy” paragraphs. Long paragraphs with short words tend to be “stretchier”

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

- When running under `luametaTEX`, the log may be filled with lines like “`luatex warning > tex: left parfill skip is gone`”. This is harmless and can be ignored.
- Lua-widow-control will not expand the first paragraph of a document. This should never be an issue in any real-world documents.

THE ALGORITHM

Lua-widow-control uses a fairly simple algorithm to eliminate widows and orphans. It is pretty basic, but there are a few subtleties. Please see “[Implementation](#)” for a full listing of the source code.

Paragraph Breaking

First, lua-widow-control hooks into the paragraph breaking process.

Before a paragraph is broken by `TEX`, lua-widow-control grabs the unbroken paragraph. Lua-widow-control then breaks the paragraph 1 line longer than its natural length and stores it for later.

Lua-widow-control does not interfere with how `TEX` breaks paragraphs into their natural length.

After `TEX` has broken its paragraph into its natural length, lua-widow-control appears once again. Before the broken paragraph is added to the main vertical list, lua-widow-control tags the first and last nodes of the paragraph. These tags create a relationship between the previously-saved lengthened paragraph and the start/end of the naturally-typeset paragraph on the page.

Page Breaking

Lua-widow-control intercepts `\box255` after the output routine and before it is shipped out.

First, lua-widow-control analyzes the `\outputpenalty` of the page. If the page was broken at a widow or orphan, the `\outputpenalty` will have been set to the respective values of `\widow\allowbreak penalty` and `\orphan\allowbreak penalty`. If the `\outputpenalty` is not indicative of a widow or orphan, lua-widow-control will abort and return `\box255` unmodified.

At this point, we know that we have a widow or orphan on the page, so we know that we need to lengthen the page by 1 line. We can iterate through the list of saved paragraphs to find the lengthened paragraph with the least demerits. Once we've selected a paragraph to replace, we can now traverse through the page to

find the original version of this paragraph so that we can splice in the lengthened paragraph.

Finally, we pull the last line off of the page and then we place it onto the top of the *recent contributions* list. Now, we can ship out the new, widow-free page.

CONTRIBUTIONS

If you have any issues with lua-widow-control, please create an issue at the [project's GitHub page](#). Or, if you are willing to help with any improvements, [submit a PR](#).

LICENSE

Lua-widow-control is licensed under the [Mozilla Public License, version 2.0](#) or greater. The documentation is licensed under [CC-BY-SA, version 4.0](#) or greater as well as the MPL.

REFERENCES

1. Knuth, DE (2020). *The T_EXBook*. Addison–Wesley. <https://ctan.org/pkg/texbook>
2. Eijkhout, V (2007). *T_EXby Topic*. Author. <https://texdoc.org/serve/texbytopic/o>
3. Isambert, P (2010). Strategies against widows. *TUGboat*, 31(1), 12–17. <https://www.tug.org/TUGboat/tb31-1/tb97isambert.pdf>
4. Mittelbach, F (2018). Managing forlorn paragraph lines in L^AT_EX. *TUGboat*, 39(3), 246–251. <https://www.tug.org/TUGboat/tb39-3/tb123mitt-widows.pdf>
5. jeremie (2017, August). *Paragraph callback to help with widows/orphans hand tuning*. <https://tex.stackexchange.com/q/372062>
6. Mittelbach, F (2021, March). *The widows-and-orphans package*. Author. <https://www.ctan.org/pkg/widows-and-orphans>

IMPLEMENTATION

lua-widow-control.lua

```
--[[
    lua-widow-control
    https://github.com/gucci-on-fleek/lua-widow-control
    SPDX-License-Identifier: MPL-2.0+
    SPDX-FileCopyrightText: 2021 gucci-on-fleek
]]

lwc = {}
lwc.name = "lua-widow-control"

--[[
    lua-widow-control is intended to be format-agnostic. It only runs on LuaTEX,
    but there are still some slight differences between formats. Here, we
    detect the format name then set some flags for later processing.
]]
local format = tex.formatname

if format:find('cont') then -- cont-en, cont-fr, cont-nl, ...
    lwc.context = true
elseif format:find('latex') then -- luatex, luatex-dev, ...
    lwc.latex = true
elseif format == 'luatex' then -- Plain
    lwc.plain = true
end

--[[
    Save some local copies of the node library to reduce table lookups.
    This is probably a useless micro-optimization, but it can't hurt.
]]
local last = node.slide
local copy = node.copy_list
local par_id = node.id("par")
local glue_id = node.id("glue")
local set_attribute = node.set_attribute
local has_attribute = node.has_attribute
local flush_list = node.flush_list or node.flushlist
local min_col_width = tex.sp("25em")
local maxdimen = 1073741823 -- \maxdimen in sp

--[[
```

```

    This error is raised in the following circumstances:
    - When the user manually loads the Lua module without loading LuaTeXBase
    - When the package is used with an unsupported format
    Both of these are pretty unlikely, but it can't hurt to check.
]]
assert(lwc.context or luatexbase, [[

    This module requires a supported callback library. Please
    follow the following format-dependant instructions:
    - LaTeX: Use a version built after 2015-01-01, or include
        '\usepackage{luatexbase}' before loading this module.
    - Plain: Include '\input ltluatex' before loading this module.
    - ConTeXt: Use the LMTX version.
]])

--[[
    Package/module initialization
]]
if lwc.context then
    lwc.warning = logs.reporter("module", lwc.name)
    lwc.attribute = attributes.public(lwc.name)
    lwc.contrib_head = 'contribute_head' -- For luametaTeX
elseif lwc.plain or lwc.latex then
    luatexbase.provides_module {
        name = lwc.name,
        date = "2021/10/09", --%%date
        version = "1.0.0", --%%version
        description = [[

            This module provides a LuaTeX-based solution to prevent
            widows and orphans from appearing in a document. It does
            so by increasing or decreasing the lengths of previous
            paragraphs.

            ]],
        }
    lwc.warning = function(str) luatexbase.module_warning(lwc.name, str) end
    lwc.attribute = luatexbase.new_attribute(lwc.name)
    lwc.contrib_head = 'contrib_head' -- For LuaTeX
else -- uh oh
    error("Unsupported format. Please use (Lua)LaTeX, Plain (Lua)TeX, or ConTeXt
(MKXL/LMTX)")
end

lwc.paragraphs = {} -- List to hold the alternate paragraph versions

```

```

if tex.interlinepenalty ~= 0 then
    lwc.warning [[
\interlinepenalty is set to a non-zero value.
This may prevent lua-widow-control from
properly functioning.
]]
end

--[[
    Function definitions
]]

--- Create a table of functions to enable or disable a given callback
--- @param t table Parameters of the callback to create
---     callback: str = The LuaTeX callback name
---     func: function = The function to call
---     name: str = The name/ID of the callback
---     category: str = The category for a ConTeXt "Action"
---     position: str = The "position" for a ConTeXt "Action"
---     lowlevel: bool = If we should use a lowlevel LuaTeX callback instead of a
---                       ConTeXt "Action"
--- @return table t Enablers/Disablers for the callback
---     enable: function = Enable the callback
---     disable: function = Disable the callback
function lwc.register_callback(t)
    if lwc.plain or lwc.latex then -- Both use LuaTeXBase for callbacks
        return {
            enable = function()
                luatexbase.add_to_callback(t.callback, t.func, t.name)
            end,
            disable = function()
                luatexbase.remove_from_callback(t.callback, t.name)
            end,
        }
    elseif lwc.context and not t.lowlevel then
        return {
            -- Register the callback when the table is created,
            -- but activate it when `enable()` is called.
            enable = nodes.tasks.appendaction(t.category, t.position, "lwc." .. t.name)
                or function()
                    nodes.tasks.enableaction(t.category, "lwc." .. t.name)
                end,
        }
    end,
end,

```

```

        disable = function()
            nodes.tasks.disableaction(t.category, "lwc." .. t.name)
        end,
    }
elseif lwc.context and t.lowlevel then
    --[[
        Some of the callbacks in ConTEXt have no associated "actions". Unlike
        with LuaTEXbase, ConTEXt leaves some LuaTEX callbacks unregistered
        and unfrozen. Because of this, we need to register some callbacks at the
        engine level. This is fragile though, because a future ConTEXt update
        may decide to register one of these functions, in which case
        lua-widow-control will crash with a cryptic error message.
    ]]
    return {
        enable = function() callback.register(t.callback, t.func) end,
        disable = function() callback.register(t.callback, nil) end,
    }
end
end

--- Saves each paragraph, but lengthened by 1 line
function lwc.save_paragraphs(head)
    -- Prevent the "underfull hbox" warnings when we store a potential paragraph
    lwc.callbacks.disable_box_warnings.enable()

    -- Ensure that we were actually given a par (only under ConTEXt for some reason)
    if head.id ~= par_id and lwc.context then
        return head
    end

    -- We need to return the unmodified head at the end, so we make a copy here
    local new_head = copy(head)

    -- Prevent ultra-short last lines (TEXBook p. 104), except with narrow columns
    local parfillskip = last(new_head)
    if parfillskip.id == glue_id and tex.hsize > min_col_width then
        parfillskip.stretch_order = 0
        parfillskip.stretch = 0.9 * tex.hsize -- Last line must be at least 10% long
    end

    -- Break the paragraph 1 line longer than natural
    local long_node, long_info = tex.linebreak(new_head, {
        looseness = 1,
    })

```

```

    emergencystretch = tex.dimen.lwcemergencystretch,
  })

  -- Break the natural paragraph so we know how long it was
  local natural_node, natural_info = tex.linebreak(copy(head))
  flush_list(natural_node)

  lwc.callbacks.disable_box_warnings.disable()

  -- If we can't lengthen the paragraph, assign a very large demerit value
  local long_demerits
  if long_info.prevgraf == natural_info.prevgraf then
    long_demerits = maxdimen
  else
    long_demerits = long_info.demerits
  end

  -- Offset the accumulated \prevdepth
  local prevdepth = node.new("glue")
  prevdepth.width = -1 * long_info.prevdepth
  last(long_node).next = prevdepth

  table.insert(lwc.paragraphs, {demerits = long_demerits, node = long_node})

  -- luametaTeX crashes if we return `true`
  return head
end

--- Tags the beginning and the end of each paragraph as it is added to the page.
---
--- We add an attribute to the first and last node of each paragraph. The ID is
--- some arbitrary number for lua-widow-control, and the value corresponds to the
--- paragraphs index, which is negated for the end of the paragraph.
function lwc.mark_paragraphs(head)
  set_attribute(head, lwc.attribute, #lwc.paragraphs)
  set_attribute(last(head), lwc.attribute, -1 * #lwc.paragraphs)

  return head
end

--- Remove the widows and orphans from the page, just after the output routine.
---
```

```

--- This function holds the "meat" of the module. It is called just after the
--- end of the output routine, before the page is shipped out. If the output
--- penalty indicates that the page was broken at a widow or an orphan, we
--- replace one paragraph with the same paragraph, but lengthened by one line.
--- Then, we can push the bottom line of the page to the next page.
function lwc.remove_widows(head)
    local penalty = tex.outputpenalty
    local paragraphs = lwc.paragraphs

    --[[
        We only need to process pages that have orphans or widows. If `paragraphs`
        is empty, then there is nothing that we can do.
    ]]
    if penalty >= 10000 or
        penalty <= -10000 or
        penalty == 0 or
        #paragraphs == 0 then
        return head
    end

    local head_save = head -- Save the start of the `head` linked-list

    --[[
        Find the paragraph on the page with the minimum penalty.

        This would be a 1-liner in Python or JavaScript, but Lua is pretty low-level,
        so there's quite a bit of code here.
    ]]
    local paragraph_index = 1
    local minimum_demerits = paragraphs[paragraph_index].demerits

    -- We find the current "best" replacement, then free the unused ones
    for i, paragraph in pairs(paragraphs) do
        if paragraph.demerits < minimum_demerits and i <= #paragraphs - 1 then
            flush_list(paragraphs[paragraph_index].node)
            paragraphs[paragraph_index].node = nil
            paragraph_index, minimum_demerits = i, paragraph.demerits
        elseif i > 1 then
            -- Not sure why `i > 1` is required?
            flush_list(paragraph.node)
            paragraph.node = nil
        end
    end
end

```



```

local target_node = paragraphs[paragraph_index].node
local clear_flag = false

-- Loop through all of the nodes on the page
while head do
  -- Insert the start of the replacement paragraph
  if has_attribute(head, lwc.attribute, paragraph_index) then
    head.prev.next = target_node
    clear_flag = true
  end

  -- Insert the end of the replacement paragraph
  if has_attribute(head, lwc.attribute, -1 * paragraph_index) then
    last(target_node).next = head.next
    clear_flag = false
  end

  -- Start of final paragraph
  if has_attribute(head, lwc.attribute, #paragraphs) then
    local last_line = copy(last(head))

    last(last_line).next = copy(tex.lists[lwc.contrib_head])

    last(head).prev.prev.next = nil
    -- Move the last line to the next page
    tex.lists[lwc.contrib_head] = last_line
  end

  if clear_flag then
    head = node.free(head)
  else
    head = head.next
  end
end

lwc.paragraphs = {} -- Clear paragraphs array at the end of the page

return head_save
end

-- Add all of the callbacks
lwc.callbacks = {
  disable_box_warnings = lwc.register_callback({

```

```

        callback = "hpack_quality",
        func      = function() end,
        name      = "disable_box_warnings",
        lowlevel  = true,
    },
    remove_widows = lwc.register_callback({
        callback = "pre_output_filter",
        func     = lwc.remove_widows,
        name     = "remove_widows",
        lowlevel = true,
    }),
    save_paragraphs = lwc.register_callback({
        callback = "pre_linebreak_filter",
        func     = lwc.save_paragraphs,
        name     = "save_paragraphs",
        category = "processors",
        position = "after",
    }),
    mark_paragraphs = lwc.register_callback({
        callback = "post_linebreak_filter",
        func     = lwc.mark_paragraphs,
        name     = "mark_paragraphs",
        category = "finalizers",
        position = "after",
    }),
}

```

```

local enabled = false
function lwc.enable_callbacks()
    if not enabled then
        lwc.callbacks.remove_widows.enable()
        lwc.callbacks.save_paragraphs.enable()
        lwc.callbacks.mark_paragraphs.enable()

        enabled = true
    else
        lwc.warning("Already enabled")
    end
end

```

```

function lwc.disable_callbacks()
    if enabled then

```

```

lwc.callbacks.save_paragraphs.disable()
lwc.callbacks.mark_paragraphs.disable()
--[
    We do not disable 'remove_widows' callback, since we still want
    to expand any of the previously-saved paragraphs if we hit an orphan
    or a widow.
]]

enabled = false
else
    lwc.warning("Already disabled")
end
end
end

return lwc

lua-widow-control.tex
% lua-widow-control
% https://github.com/gucci-on-fleek/lua-widow-control
% SPDX-License-Identifier: MPL-2.0+
% SPDX-FileCopyrightText: 2021 gucci-on-fleek

\wlog{lua-widow-control v1.0.0} %%version

\input ltluatex % LuaTeXBase

\clubpenalty=1
\widowpenalty=1
\displaywidowpenalty=0
\interlinepenalty=0
\brokenpenalty=0

\newdimen\lwcemergencystretch
\lwcemergencystretch=3em

\directlua{require "lua-widow-control"}

% Here, we enable font expansion/contraction. It isn't strictly necessary for
% lua-widow-control's functionality; however, it is required for the
% lengthened paragraphs to not have terrible spacing.
\expandglyphsinfont\the\font 20 20 5
\adjustspacing=2

```

```

\def\lwcenable{\directlua{lwc.enable_callbacks()}}
\def\lwcdisable{\directlua{lwc.disable_callbacks()}}

% Enable lua-widow-control by default when the package is loaded.
\lwcenable

\endinput

lua-widow-control.sty
% lua-widow-control
% https://github.com/gucci-on-fleek/lua-widow-control
% SPDX-License-Identifier: MPL-2.0+
% SPDX-FileCopyrightText: 2021 gucci-on-fleek

\NeedsTeXFormat{LaTeX2e}[2015/01/01] % Formats built after 2015 include LuaTeXBase
\ProvidesPackage{lua-widow-control}%
    [2021/10/09 v1.0.0] %%version %%date

\setlength{\clubpenalty}{1}
\setlength{\widowpenalty}{1}
\setlength{\displaywidowpenalty}{0}
\setlength{\interlinepenalty}{0}
\setlength{\brokenpenalty}{0}

% We can't use \newlength since that makes a TeX "skip", not a "dimen"
\newdimen\lwcemergencystretch
\setlength{\lwcemergencystretch}{3em}

\directlua{require "lua-widow-control"}

% Here, we enable font expansion/contraction. It isn't strictly necessary for
% lua-widow-control's functionality; however, it is required for the
% lengthened paragraphs to not have terrible spacing.
\RequirePackage[final]{microtype}

\newcommand{\lwcenable}{\directlua{lwc.enable_callbacks()}}
\newcommand{\lwcdisable}{\directlua{lwc.disable_callbacks()}}

% Enable lua-widow-control by default when the package is loaded.
\lwcenable

\endinput

```

t-lua-widow-control.mkx

```
%D \module
%D   [      file=t-lua-widow-control,
%D       version=1.0.0, %%version
%D       title=lua-widow-control,
%D       subtitle=ConTeXt module for lua-widow-control,
%D       author=gucci-on-fleek,
%D       date=2021-10-09, %%date
%D       copyright=gucci-on-fleek,
%D       license=MPL-2.0+,
%D       url=https://github.com/gucci-on-fleek/lua-widow-control]
\startmodule[lua-widow-control]
\unprotect

\installnamespace{lwc}

\installcommandhandler \????lwc {lwc} \????lwc

\newdimen\lwcemergencystretch
\appendtoks
  \lwcemergencystretch=\lwcparameter{emergencystretch}
\to\everysetuplwc

\appendtoks
  \doifelse{\lwcparameter{\c!state}}{\v!start{
    \ctxlua{lwc.enable_callbacks()}}
  }{
    \ctxlua{lwc.disable_callbacks()}}
  }
\to\everysetuplwc

\ctxloadluafile{lua-widow-control}

\setuplwc[emergencystretch=3em, \c!state=\v!start]

% We can't just set the penalties because they will be reset automatically
% at \starttext.
\startsetups[*default]
  \clubpenalty=1
  \widowpenalty=1
  \displaywidowpenalty=0
  \interlinepenalty=0
  \brokenpenalty=0
\stopsetups
```

```
\setups[*default]
```

```
% Here, we enable font expansion/contraction. It isn't strictly necessary for  
% lua-widow-control's functionality; however, it is required for the  
% lengthened paragraphs to not have terrible spacing.
```

```
\definefontfeature[default][default][expansion=quality]
```

```
\setupalign[hz]
```

```
\protect
```

```
\stopmodule
```

Demo from Table 1

```
\definepapersize[smallpaper][
```

```
    width=6cm,
```

```
    height=9cm
```

```
]\setuppapersize[smallpaper]
```

```
\setuplayout[
```

```
    topspace=0.1cm,
```

```
    backspace=0.1cm,
```

```
    width=middle,
```

```
    height=middle,
```

```
    header=0pt,
```

```
    footer=0pt,
```

```
]
```

```
\def\lwc/{\sans{lua-\allowbreak widow-\allowbreak control}}
```

```
\def\Lwc/{\sans{Lua-\allowbreak widow-\allowbreak control}}
```

```
\setupbodyfont[9pt]
```

```
\setupindenting[yes, 2em]
```

```
\setupalign[tolerant]
```

```
\definepalet[layout][grid=middlegray]
```

```
\showgrid[nonumber, none, lines]
```

```
\definefontfeature[default][default][expansion=quality]
```

```
\setupalign[hz]
```

```
\starttext
```

`\Lwc/` can remove most widows and orphans from a document, `\emph{without}` stretching any glue or shortening any pages.

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, `\lwc/` is breaking the paragraph 1~line longer than its natural length. `\TeX{}{}{}{}`'s paragraph is output to the page, but `\lwc/`'s paragraph is just stored for later. When a widow or orphan occurs, `\lwc/` can take over. It selects the previously-saved paragraph with the least badness; then, it replaces `\TeX{}{}{}{}`'s paragraph with its saved paragraph. This lengthens the page by 1~line.

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 for the document's author.

`\stoptext`