The `fewerfloatpages` package*

Frank Mittelbach

Abstract

LaTeX's float algorithm has the tendency to produce fairly empty float pages, i.e., pages containing only floats but with a lot of free space remaining that could easily be filled with nearby text. There are good reasons for this behavior; nevertheless, the results look unappealing and in many cases documents are unnecessarily enlarged.

The fewerfloatpages package provides an extended algorithm that improves on this behavior without the need for manual intervention by the user.

Contents

1 Introduction

We start by giving a quick overview of LaTeX's float algorithm and the problems that result from the approach used.

We then look in some detail into possible alterations and improvements to that algorithm and discuss possible issues that need to be resolved. In this section we also describe all configuration possibilities of the extended algorithm.

The final section then documents the code changes that are necessary to LaTeX kernel macros to implement the extension.

1.1 A quick overview of LaTeX's float algorithm

LaTeX's output routine uses a greedy algorithm to place floats near to their call-outs in the source document. The decision of how to place a float is made when the float is first encountered. If possible it is placed onto the current page, either in mid-text, on top or into the bottom area, depending on what is allowed for the float and how many floats are already placed into those areas.

If the float can't be placed immediately, it goes into a defer list, and in order to not accumulate too many unplaced floats LaTeX tries to empty that list whenever there is a chance. This chance comes after the next page break: LaTeX then starts a special "float page" algorithm in which it examines the defer list and from it forms float pages (i.e., pages that contain only floats). If necessary, it generates several float pages and only stops if there are no floats waiting to be placed, or there are too few floats to form a float page, or there are only floats left that are for one or another reason not allowed to be placed in this way.

Finally LaTeX looks at the remaining floats and tries to place as many of them as possible into the top and bottom area of the next page. Then it continues to process further text to fill the text part of that page. Details on the exact behavior of the algorithm are discussed in [1].

* The current package version is `v1.0b` dated 2021/03/02.

1.2 The typical float page and its problems

LaTeX considers a float page to be successfully built if its floats take up more than `\floatpagefraction` of the whole page. By default this parameter is set to `.5` which means that such float pages may end up being half empty.

Many users think that this is not a good value and try to improve on it by enforcing a higher percentage (such as 80%) only to find that this prevents LaTeX in many cases from successfully generating any float page, with the effect that all floats are suddenly piling up at the end of the document.

Why is this the case? In a nutshell, because a higher percentage makes it much more likely that a float can't be placed, because it is not big enough to be used on its own and no other nearby floats can be combined with it, because their combination violates some other restriction, e.g., together they are bigger than a page, not all of them are allowed to go on float pages, etc. The moment that happens this float prevents the placement of all later floats of the same class too (i.e., all figures) and disaster is ensured. In most cases these floats will then never get placed, because they need a float of the right size from a different class to appear, which may in theory happen but is, unfortunately, unlikely.

Tinkering with the parameter settings will usually produce unwanted effects

Thus, while tempting, tinkering with this parameter by making it larger is usually not a good idea, unless you are prepared to place most if not all of your floats manually, by overwriting the placement algorithm on the level of individual floats (e.g., using `!` syntax and/or shifting its position in the source document).

Why does the current algorithm have these problems? To some extent, because it offers only global parameters that need to fit different scenarios and thus settings that are suitable when many floats need to be placed result in sub-optimal paginations in document parts that contain only a few floats, and vice versa. To overcome this problem, either one can try to develop algorithms with many more configurable parameters that act differently in different scenarios or one can let the algorithm follow a main strategy, configurable with only a few parameters (like today), but monitor the process and make more local adjustments and corrections depending on the actual outcome of that base strategy and additional knowledge of the actual situation in a given document part. This is the approach taken by the extension implemented in this package.

2 Improvements to the float page algorithm

A simple way to improve on the existing algorithm, without compromising its main goal of placing the floats as fast as possible and as close as possible to their call-outs, is the following: as long as there are many floats waiting to be placed, generate float pages as necessary to get them placed (using the current algorithm and its parameters).

Once we are unable to build further float pages, do some level of backtracking by checking if we have actually succeeded in placing all floats. If there are still floats waiting to be placed then assume that what has been done so far is the best possible way to place as many floats as possible (which it probably is). However, if we have been able to place all floats onto float pages then check if the last float page is sufficiently full; if not, undo that float page and instead redistribute its floats into the top and bottom area of the next upcoming page. This way the floats will be combined with further text and we avoid a possible half-empty float page.

This approach will not resolve all the problematic scenarios where we find that LaTeX has decided to favor fairly empty float pages over some tighter type of placement. It will, however, help to improve typical cases that do not involve too many floats. For a example, if a single (larger) float appears near the end of a page, then using the standard algorithm it can't be immediately placed (because there isn't enough free space on the current page). It is therefore moved to the defer list and at the page break it is then placed onto a float page (possibly by itself, if it is large enough to allow

A typical case where we don't really want LaTeX to make a float page

for that) even though it could perfectly well go into the top or bottom area of the next page and thus be combined with textual material on that page.

With the new algorithm this float page is reexamined and unless it is pretty much filled up already, it is unraveled and its floats are redistributed into the top and bottom areas of the next page. If, however, we have many floats waiting on the defer list, the normal float page algorithm will first place as many of them as possible into float pages and only the last of these pages will be subject to a closer inspection and a possible unraveling.

An extension of this idea (and the one that we actually implement) is to monitor the whole float page generation process and instead of just considering the last float page in the sequence for unraveling, we look at each prospective float page in turn and based on the current situation (e.g., number of floats still being unplaced, free space on the float page, etc.) decide whether this float page should be produced or whether we should stop making float pages and instead place the pending floats into top and bottom areas of the upcoming page.

2.1 Details on the extended algorithm

Don't unravel a float page if there are too many floats on the defer list

The main idea of the extended algorithm is to avoid unnecessary cases of float pages especially if those float pages are fairly empty. Natural candidates are single float pages, but even in cases where the current LaTeX algorithm produces several sequential float pages the extended algorithm may decide to replace them by normal pages under certain conditions. However, the main goal is and should remain to place as many floats as soon as possible and so generating float pages when many floats are waiting is usually essential.

`floatpagedeferlimit` `\setcounter{floatpagedeferlimit}{`⟨*number*⟩`}`

Whether or not unraveling for a float page is considered at all is guided by the counter `floatpagedeferlimit`. As long as there are more floats waiting on the defer list than this number, float pages are not considered for unraveling. The default is `3` which corresponds to the default value for `totalnumber`, i.e., with that setting the unraveling of a floating page has a fighting chance to place all floats into the top and bottom areas on the current page. It would also resolve cases for up to three floats, each larger than `\floatpagefraction`, where the standard LaTeX algorithm would produce three individual float pages.

If you set the counter to `1` then only the last float page in a sequence is considered, and only if it contains only a single float and if there are no other floats that are still waiting to be placed. If you set it to `0`, then the extension is disabled, because float pages are produced only if there was at least one float on the defer list.

Don't unravel if the float page contains many floats

Even if we set `floatpagedeferlimit` to a fairly high value, we may not want to unravel float pages that contain many floats. To support this case there is a second counter that guides the algorithm in this respect.

`floatpagekeeplimit` `\setcounter{floatpagekeeplimit}{`⟨*number*⟩`}`

Whenever the float page contains at least `floatpagekeeplimit` floats it will not be unraveled. The default is also `3` so that float pages with three or more floats are not touched. Obviously the counter can have any effect only if it has a value less than or equal to `floatpagedeferlimit` because this is tested first.

Don't unravel if the float page contains at least one `[p]` *float*

There are, however, a number of other situations in which we shouldn't unravel a float page even if the above checks for the size of the defer list were passed successfully. The most important one is the case when the float page contains at least one float that is allowed *only* on float pages (i.e., has a `[p]` argument). Such a float would not be

placeable in a top/bottom area on any page and thus would be repeatedly sent back to the defer list (possibly forever).

Don't unravel if the float page is nearly filled

The other case where unraveling would normally be counterproductive is when the particular float page is nearly or completely filled up with floats. If we unravel it, then it is certain that we can place only some of the floats into the top or bottom area of the next page, while some would end up on the defer list. That in turn means that these deferred floats float even further away from their call-out positions than need be.

`\floatpagekeepfraction` `\renewcommand\floatpagekeepfraction{`⟨*decimal*⟩`}`

So what is a good way to determine if a float page is "full enough"? A possible answer is that if the remaining free space on that page is less than `\textfraction` we consider it full enough to stay. `\textfraction` defines the minimum amount of space that has to be occupied by text on a normal page, thus if all floats together need so much space that this amount of text could not fit, then trying to place all floats onto a normal page can't succeed and some of them would get deferred for sure. To allow for further flexibility the algorithm uses the variable `\floatpagekeepfraction` (defaulting to `\textfraction`) so if desired a lower (or even a higher) boundary can be set.

The above parameters give some reasonable configuration possibilities to guide the algorithm as to when and when not to unravel a possible float page and instead produce further normal pages. It should be noted, however, that except for the case of setting `floatpagedeferlimit` to `1`, there is always a chance that floats drift further away from their call-outs, because they may not be immediately placeable due to other parameter settings of the float algorithm. For example, the counter `topnumber` (default value 2) limits the number of floats that can be placed in the top area on a normal page and if more remain after unraveling only two can immediately go in this area.

2.2 Possible pitfalls and how to avoid them

The algorithm detects if a float is allowed only on float pages (i.e., is given in the source as `[p]`) and it will ensure that float pages containing such floats are not unraveled.

However, if you have a float with the default specifier `[tbp]` whose size is larger than the allowed size of the top or bottom area (e.g., larger than `\topfraction` × `\textheight`), then this effectively means it can only be placed on a float page.

However, according to the specifier the float is allowed to go into the top or bottom area, so the algorithm, as explained so far, would be allowed to unravel and when that float later is considered for top or bottom placement it will get again deferred and thus move from one page to the next, most likely messing up the whole float placement.

`checktb` *(option)*

There are two possible ways to improve the algorithm to avoid this disaster. One way would be to check the float size when it is initially encountered and remove any specifier that is technically not possible because of the parameter settings and the float size. A possible disadvantage is that this determination will be done once and any later (temporary) change to the float parameters will have no effect. This is currently the package default. It can be explicitly selected by specifying the option `checktb`. In this case you might see warnings like

```
LaTeX Warning: Float too large for top area: t changed to p on line ...
```

`addbang` *(option)*

Another possibility is that we automatically add a `!` specifier to all floats during unraveling, i.e., when we send them back for reevaluation. This way such floats become placeable into top and bottom areas regardless of their size. This may result in fewer pages at the cost of violating the area size restrictions once in a while. It is specified with the option `addbang`.

`nocheck` *(option)*

If you prefer no automatic adjustment of the specifiers, add the option `nocheck`. In this case you might find that floats of certain sizes are unplaceable and thus get delayed

to the end of the document. If that happens, the remedy is either to explicitly specify `[p]` or `[hp]` for such a float (to ensure that they aren't subject to unraveling) or to manually add an exclamation specifier, e.g., `[!tp]` so that LaTeX doesn't use the size restrictions in its algorithm.

2.3 Tracing the algorithm

`trace` *(option)* The package offers the option `trace`, which if used, will result in messages such as

```
[1]
fewerfloatpages: PAGE: trying to make a float page
fewerfloatpages: ----- \@deferlist: \bx@B \bx@D
fewerfloatpages:  starting with \bx@B
fewerfloatpages:  --> success:  \bx@B \bx@D
fewerfloatpages: ----- current float page unraveled
                (free space 192.50336pt > 109.99832pt)
[2]
```

which means that the algorithm is trying to make a float page from the defer list which at that point contained two floats (the float boxes `\bx@B` and `\bx@D`), that it was able to produce a float page containing just `\bx@B` and `\bx@D`, and that the extended algorithm then decided to unravel that float page, because it has an unused space of `192.5pt`, i.e., roughly 16 text lines. With the current `\floatpagekeepfraction` that is too much empty space on the page.

Or it might say

```
fewerfloatpages: PAGE: trying to make a float page
fewerfloatpages: ----- \@deferlist: \bx@D \bx@F \bx@G \bx@H \bx@I
fewerfloatpages:  starting with \bx@D
fewerfloatpages:  --> success: \bx@D \bx@F
fewerfloatpages: ----- too many deferred floats for unraveling (5 > 3)
[3]
```

which means that the algorithm made a float page out of the first two floats from the defer list (i.e., 3 remained). That page was kept regardless of the amount of free space it contained because we have a total of 5 floats on the defer list and the counter `floatpagedeferlimit` has its default value of `3`.

The above tracing messages are both from the same test document. What they also (implicitly) show is that the unraveling that happened after page 1 resulted in only one float (`\bx@B`) being placed on page 2, because we see the second one (`\bx@D`) reappearing in the defer list after page 2 got finished. In other words it was moved one page further away from its call-out: the price for getting a nicely filled page 2 instead of a fairly empty float page with roughly 200 points left empty. The final part of that test document then exhibits another type of message:

```
fewerfloatpages: PAGE: trying to make a float page
fewerfloatpages: ----- \@deferlist: \bx@G \bx@H \bx@I
fewerfloatpages:  starting with \bx@G
fewerfloatpages:  --> success: \bx@G \bx@H \bx@I
fewerfloatpages: ----- all floats placed on float page(s)
fewerfloatpages: ----- current float page kept, full enough
                      (free space 38.99496pt < 109.99832pt)
[4]
```

This means that the remaining floats (that were left unplaced after float page 3 got constructed) formed a float page and that float page was the last in sequence (i.e., all floats have been placed). However, this time the algorithm decided not to unravel it, because it is nicely full: there are only 39 points of free space left on that page.

Three other possible messages are shown in this sequence of tracing lines from a second test document (which is using some uncommon settings: `floatpagedeferlimit` is `10` and `floatpagekeeplimit` is `5`):

```
[1]
fewerfloatpages: PAGE: trying to make a float page
fewerfloatpages: ----- \@deferlist: \bx@B \bx@C \bx@D \bx@E \bx@F \bx@G \bx@H
fewerfloatpages:  starting with \bx@B
fewerfloatpages:  --> success: \bx@B \bx@C \bx@D \bx@E \bx@F \bx@G \bx@H
fewerfloatpages: ----- current float page kept (contains at least 5 floats)
[2] [3]
```

In this case 7 floats have been waiting on the defer list and the algorithm was able to construct a float page using all of them. The algorithm then keeps that page because it has 5 or more floats in it (the value of the `floatpagekeeplimit` counter).

The next message in that test document shows what happens when there are not enough floats waiting or they are simply too small (to even get past the `\floatpagefraction` limit):

```
fewerfloatpages: PAGE: trying to make a float page
fewerfloatpages: ----- \@deferlist: \bx@I \bx@J
fewerfloatpages:  starting with \bx@I
fewerfloatpages:  --> fail
fewerfloatpages:  starting with \bx@J
fewerfloatpages:  --> fail
fewerfloatpages:  --> fail: no float page made
[4]
```

So no float page was made, but for some reason (that becomes clear later) the two floats also didn't got distributed into the top or bottom area of the next page. Instead they remained on the defer list and during processing of page 4 one more float was found so that after that page the defer list had grown to length 3:

```
fewerfloatpages: PAGE: trying to make a float page
fewerfloatpages: ----- \@deferlist: \bx@I \bx@J \bx@K
fewerfloatpages:  starting with \bx@I
fewerfloatpages:  --> success: \bx@I \bx@J \bx@K
fewerfloatpages: ----- current float page kept, contains a float
fewerfloatpages:       with p but no t or b specifier
[5]
```

This time all floats could be placed, but again the float page wasn't unraveled (even though in the test document it contained a lot of white space) because of the fact that one of its floats (in fact the first though that can only be deduced implicitly) was specified as a "float page only" float. This explains why on page 4 `\bx@I` couldn't be placed into the top or bottom area and then all following floats of the same class (the test document contained only `figure` floats) couldn't be placed either.

Detailed tracing of the complete algorithm

If you want detailed tracing of the complete algorithm, also load the fltrace package and enable the tracing with `\tracefloats` anywhere in your document. Note, however, that the resulting output is very detailed but rather low-level and unpolished.

2.4 Local (manual) adjustments

If the extended algorithm is used you will get fewer float pages that contain a noticeable amount of white space. By adjusting `\floatpagekeepfraction` and the counters `floatpagekeeplimit` and `floatpagedeferlimit` you can direct the algorithm to unravel more or fewer of the otherwise generated float pages. However, in some cases it might happen that redistribution of the floats into the top and bottom areas of the

next page(s) may result in some of them drifting too far away from their call-outs. If that happens, you can either try to change the general parameters or you could help the algorithm along by using the optional argument of individual float environments. The two main tools at your disposal are

- using the `[!..]` notation to allow the float to go into the top or bottom area even if it would be normally prevented by other restrictions;
- using `[p]` to force a float into a float page as that prevents the algorithm from unravelling the float page which contains that float.

As an alternative you can, of course, temporarily alter the definition of the command `\floatpagekeepfraction` or the values of two counters in mid-document, but remember that they are not looked at when a float is encountered in the source but when we are at a page break and LaTeX attempts to empty the defer list, which is usually later and unfortunately somewhat asynchronous, i.e., not easy to predict.

References

[1] Frank Mittelbach. How to influence the position of float environments like figure and table in LaTeX? *TUG*boat 35:3, 2014.
`https://www.latex-project.org/publications/indexbytopic/2e-floats/`

[2] LaTeX Project Team. The LaTeX 2_ε Sources (660+ pages), 2020.
`https://www.latex-project.org/help/documentation`

Index

◇ Frank Mittelbach
Mainz, Germany
`https://www.latex-project.org`
`https://ctan.org/pkg/fewerfloatpages`