Formatting documents with floats
A new algorithm for LaTeX 2$_\varepsilon$*

Frank Mittelbach
LaTeX3 Project
frank.mittelbach@latex-project.org

Abstract

This paper describes an approach to placement of floats in multicolumn documents.

The current version of LaTeX was originally written for single-column documents and extended to support two-column documents by essentially building each column independently from the other. As a result the current system shows severe limitations in two column mode, such as the fact that spanning floats are always deferred to at least the next page or that numbering between column floats and spanning floats can get out of sequence.

The new algorithm is intended to overcome these limitations and at the same time extend the supported class of document layouts to multiple columns with floats spanning an arbitrary number of columns.

Editor's note

This paper describes facilities offered by the author's new algorithm for use with LaTeX; it therefore seemed appropriate that the paper itself should be typeset using an implementation of it, and the author was enthusiastic in support of the plan.

With some help from members of the LaTeX Team, we have managed to typeset all but the present page of the paper using a version of LaTeX that incorporates a prototype implementation of the new algorithm.

While customising the algorithm to produce the standard layout that readers of *TUGboat* have come to expect, the paper also exhibits the following capabilities of the new algorithm:

- Alignment of text lines throughout the article on an invisible grid.
- Support for spanning bottom floats; examples are on pages 285 and 288.
- Restriction of float placement.

The float placement restrictions selected for this article are as follows: floats have to appear after their call-outs, can only occupy bottom areas, and are not allowed there if footnotes are present in the column. This accounts, for example, for the placement of figure 1, which was moved from the second column of page 280 to the bottom of the first column of page 281.

In the lingua of the algorithm the exact specification used was:

```
float-callout-constraint = after,
float-callout-span-constraint
                           = flexible,
bottom-float-footnote-constraint
                           = forbidden,
max-float-num = 2,
area-list = {b12,b11,b21},
```

These settings are admittedly rather bizarre and were solemnly chosen by the author for illustration purposes.

In order to illustrate clearly the effect of the page layout grid alignment used throughout, on page 279 a grid of lines is superimposed; we hope this does not detract too much from your enjoyment in reading the article.

In general it should be noted that the *TUGboat* layout isn't really suited to be typeset using an underlying grid; headings at the top of the column need to drop to avoid a large gap between the heading and the following text (see page 279) and of course with a flush bottom setting you will get widows and orphans since there is no stretchability on the page.

This title page has been set using the standard (released) LaTeX output routine because the prototype implementation does not at present support switching the number of columns in the middle of the page.

Introduction

One problem with formatting documents containing floats is the number of potential formatting solutions that need to be checked out. The number of trials grows combinatorially in the number of floats and areas which can receive them. If we have n floats waiting to be placed and m areas in which we can place them on the current page being built (not counting the "deferred area") then the number of different placements is given by

$$\#\text{trials} = \binom{n+m}{m} = \frac{(n+m)!}{n!\,m!} \qquad (1)$$

assuming that the order of floats has to be preserved, i.e., if the call-out of float f_i is before the call-out of f_j in the text stream then the float f_i will be placed earlier than float f_j where "earlier" is a defined relation of float areas.

For example, if we have 8 floats waiting to be distributed among 12 areas (which corresponds to a three column page with float areas at the top and bottom allowing for partial spans) then we have to check 125970 possible distributions; if two additional floats appear we end up with 646646 trials.

Even though a large number of these distributions would be unacceptable and discardable straight away, after some initial test, the resulting running time of the algorithm would clearly be beyond any acceptable speed. (Assuming we could do 1000 trials per second, which is ridiculously high since many of them would require trial-typesetting the whole page, then the case of 646646 trials would still take roughly 10 minutes to form a decision.)

Thus it is important to find algorithms with complexity that is at worst linear in both the number of floats on the trial list and the number of possible float areas, even if this means that in a few cases a relatively good layout will not be found. It is even better if they have minimal redundancy.

Note that assessing the actual running time of TEX code is not straightforward since some activities are very much faster than others. For example, performing a test by using a reasonable number of macro expansions and register assignments may be very much slower than running through a long typeset list and then doing a simple test.

The algorithm we have implemented fulfills the requirement of being (essentially) linear in the number of floats and the number of float areas.

The document source model

The document source is a single stream of continuous text containing call-outs to floating objects. (At the moment the call-outs are marked by placing the objects into the stream but it would be possible to provide them as separate objects.) Floating objects (as of today) come in three incarnations:

- Objects where the call-out and the placement requires a strict spatial relationship, e.g., same line in the margin. An example would be marginal notes as implemented by `\marginpar` in LaTeX 2ε.
- Objects where the call-out and the placement are required to fall onto the same column/page/spread, e.g., footnotes.
- Objects where there is a defined relation between call-out and object placement, e.g., "not in an earlier column", or "on the same page or later", etc. These are the traditional floats.

Float objects in the last group are typed where the type is defined by the logical content of the object, e.g., "figure", "table", and so on.

The document formatting is achieved using a minimal but customizable lookahead (typically the considered galley material is the equivalent of one page/spread of textual material ignoring the additional size taken up by embedded float objects).

While making up pages the main "quality" guidance for the algorithm is to try to place each float as early as possible without violating defined constraints.

The document layout model

Page layout grid The algorithm supports the specification of a page grid on which it will align text columns and other elements. This will allow (if suitable parameters for various elements are chosen) to have text lines of different columns all lying on grid points.[1]

Columns The page layouts which are supported by the new algorithm support an arbitrary number of text columns of equal width.

The number of columns per page as well as their

[1] On the current page lines are drawn to highlight the grid. Note that headings, lists, and other "display" objects are not aligned.

width can be changed at forced page breaks such as the start of chapters.

Balanced columns Balancing columns (as done by the `multicol` package) is planned but not implemented. The major problem in that area is the handling of column floats during the balancing process.

Float areas Float objects are distributed into float areas which are rectangular in shape. Float areas span one or more text columns; their horizontal size is therefore given by the following formula (where c is the number of columns spanned):

$$\langle \textit{area-width} \rangle = c \times \big(\langle \textit{col-width} \rangle + \langle \textit{col-sep} \rangle \big) - \langle \textit{col-sep} \rangle$$

The naming conventions for float areas is as follows:

$$\langle \textit{identifier} \rangle \langle \textit{start-column} \rangle \langle \textit{span-count} \rangle .$$

The ⟨*identifier*⟩ is a single letter denoting the type of area, e.g., `t` for top, `b` for bottom. The ⟨*span-count*⟩ is a single digit denoting the number of columns to span. The ⟨*start-column*⟩ is a single digit[2] denoting the start column of the area. Thus `t23` is a top area starting at column two and spanning three columns, i.e., two, three and four. A restriction due to the naming scheme is that currently no more than 9 columns are possible.[3]

Only a subset of the float areas is allowed to be populated on a page. In essence the new algorithm does not support placements that result in "splitting" the text of a column due to a float (other than column "here" floats).[4] This means that population of some float areas must be prevented, namely those satisfying these conditions when *pcs* (where $p = \text{pos}$, $c = \text{column}$, $s = \text{span}$) has just been populated:

$$pij \text{ with } i < c \le i + j < c + s$$

or

$$pij \text{ with } i \le c + s < i + j \le \langle \textit{number-of-columns} \rangle$$

The first formula describes the areas which partly overlap from the left, the second formula describes those that partly overlap from the right. Areas which are sub- or super-areas, e.g., `t13` and `t22`, do not affect each other. The above restriction is necessary to prevent situations like the one shown in figure 1 on the facing page, i.e., where the float area `t32` (represented as b's) would result in splitting the fourth column into two independent text areas.

The possibilities, as well as the restrictions, are equal for both top and bottom areas. This means that the new scheme in particular supports spanning bottom areas.

Float pages and columns Float pages, i.e., pages consisting only of floats, will be supported as well as float columns.

Float types The type of float influences the formatting, e.g., where the caption is placed in relation to the float body, how it is formatted, what kind of fixed strings are added, etc. It also restricts the placement algorithm in respect to which float areas can be populated as explained below.

Margins The marginal areas can receive marginal notes which are aligned with the corresponding text line. In documents with more than two columns marginal notes are currently not supported though one could envision allowing them even there. If marginals have to compete for space the later marginal will be moved downwards if there is enough space on the page, otherwise the line containing the marginal will be moved to the next column/page.[5]

An alternative usage of the margin is to place footnotes into it. A prototype version of this is provided already, see section "Footnotes" on the next page.

Another potential use of the margin areas is to use them (or parts thereof) as float areas in their own right. The problem with this would be that these float areas would have a horizontal width which is different from the column width, thus allowing only a limited class of floats to appear therein.

Another potential extension would be to allow float areas that border on a margin to use the marginal space as part of the float area, thereby allowing the filling of such an area with floats which are wider than the nominal float area. A special case of this, the placement of the caption in the margin beside the float body, is already provided by choosing a suitable caption formatting instance.

[2] With a bit of care in the code this could be extended to allow more than one digit.

[3] The scheme is different from the original one used, where `t23` would have denoted an area starting at column two and spanning until column three.

[4] Perhaps this restriction will be lifted one day.

[5] This is not yet implemented — right now they overprint each other.

Footnotes Footnotes can be regarded as a special type of floats. They are objects which are associated with lines of text (their call-out) but in contrast to normal floats such as "figures" or "tables" their placement constraints are stronger, e.g., they typically have to appear at the bottom of the column which contains their call-outs, or at least they have to appear on the same page as their call-outs.

In its current version, the model supports footnotes beneath the call-out column (normal behavior); all footnotes in the last column (as with the `ftnright` package for two-column mode); all footnotes in the outer (or inner) margin.

Without an extension to the page makeup algorithm (but instead with a suitable redefinition of the footnote commands) they could be processed as marginal notes or alternatively as "end-notes".

Headers and footers The header and footer areas may use data received from individual columns. An extended version of TEX's mark mechanism is made available which allows the definition of arbitrarily many independent classes of marks. Within each mark class information about the top mark (i.e., the mark active at the top of the column), the first mark and the last mark is made available for retrieval.

This allows the production of correct running headers and footers for various types of applications such as dictionaries, manuals, etc.

The processing model

Float placement concepts To build a page or spread the algorithm first assembles enough textual material to be able to fill the page without placing any floats. During this process all floats that have their call-outs within the assembled galley are collected. They form, together with unplaced floats from previous pages, an ordered trial list of floats.

```
aaaaaaaaaaa 444
aaaaaaaaaaa 444
aaaaaaaaaaa 444
111 222
111 222 bbbbbbb
111 222 bbbbbbb
111 222 bbbbbbb
111 222
111 222 333 444
111 222 333 444
111 222 333 444
```

Figure 1: Overlapping float areas

The allowed float areas on the page under construction are totally ordered as well.

The algorithm proceeds by taking the first float from the trial list and trying to place it into the first float area from the area list. It then checks if all constraints (see below) are met and if not the algorithm will try to place the float into the next area until either all constraints are met or the areas in the float area list are exhausted. A trial that does not fail means that this distribution of floats becomes the best solution so far and all further trials will be based on adding to this solution (no backtracking). If the algorithm fails to place the float into any area it means that the float will be deferred to a later page.

As floats are added to areas, the constraints for further trials are changed. There are several reasons for this: on one hand, the call-out positions of various floats move since the float will occupy space on the page; on the other hand, placing a float in some area might result in disallowing the placement of other floats in the same or in other areas.

Float pages and columns At the moment there is only rudimentary support for float pages available: at the start of each page the algorithm will try to form a float page out of all floats that have been deferred from previous pages. However there is no layout control available to define the conditions under which such a trial will succeed.

Float storage Float bodies are typeset into boxes at the point of 'call-out', as with the `figure` and `table` environments in the standard LATEX; it may also be possible to specify at the call-out point a logical pointer to a float whose typesetting is specified elsewhere (e.g., an external file).

However, text sub-elements such as the caption, etc. (e.g., from `\caption`), are not typeset at this stage but are stored as token lists; this allows for trying different possible layout specifications, e.g., for its measure, during the float-positioning trials. At present this is confined to at most a single caption element per float.

Caption processing When a float is placed into an area the caption is trial formatted and mounted onto the float body. This process can take into account various information about the float positioning trial, such as the area to format it into, the fact that it formats onto a verso or recto page, etc. It might try several possibilities before making a decision, e.g., if

one formatting of the float results in violating some constraint(s) it might try a different formatting at this point.

Flushing floats It is possible to mark points in the source document as boundaries beyond which floats whose call-outs are prior to the boundary cannot pass. In other words a "flush point" directs the algorithm to place all affected floats into areas which are "before" the flush point.

If due to other constraints the float could not be placed in such an area the algorithm first retries all potential areas using a less rigid set of constraints (for example, restrictions on the number of allowed floats per area are dropped) and if this still doesn't enable the algorithm to place the float properly it will as a last resort move the flush point to a later column, which means breaking the column text before the flush point.

Flushing of floats can be done either for all floats or on a per float type basis, e.g., it is possible to flush only floats of type "figure".

A flush point can be given an additional attribute which controls the "fuzziness" used by the algorithm. By default the flush point algorithm uses `strict` flushing as described above. The attribute `column` modifies the algorithm's behaviour by enabling a float to move past the flush point as long as it will be placed on the same column. Similarly the attribute values `page` and `spread` will enforce that the float will not be deferred further than the current page or the current spread. This way it can be guaranteed that a float is always visible from its call-out.

Float sequence classes Float sequence classes are collections of float types; each float type belongs to exactly one float sequence class. Within each sequence class the call-out order in the document is always preserved by the float placement algorithm, e.g., if $c_1, c_2, \ldots, c_n$ are the call-outs of all floats of a float sequence class then the corresponding floats will be placed such that f_i will be placed before f_j whenever $i < j$. Thus by putting all float types into a single float sequence class all floats are placed in the order of their call-outs. At the other extreme, if each float type has its own sequence class[6] then floats from one type might move before floats of other types even though the corresponding call-outs are in a different order.

Float and call-out relations The algorithm also keeps track of the relation between an individual float and its call-out. This allows one to define constraints which guide the algorithm during the float placement phase. It is always permissible to place a float "after" its call-out, e.g., in a later column/page. At the moment the following constraints can be specified:

none which means that the relation between call-out and float placement is not relevant for placing floats.

page which means that the float can be placed anywhere on the page with the call-out (it is visible from the call-out).

column which means that the float can be placed before the call-out as long as it is placed in the same column.

after which means that the float has to be placed strictly after the call-out.

When extending the algorithm to directly support spreads the above list is going to be extended by an option that allows floats to move backwards on the whole spread.

Spanning float and call-out relations For floats that span two or more columns there are several possibilities to interpret the spatial relationship between call-out and float areas. For example, if a float, whose call-out is in the second column, has been placed into area `b12`, is this float "before" or "after" its call-out? The answer to this question depends on whether we consider the float being placed into the first or the second column, both of which are valid interpretations.

At the moment the following behaviour can be specified:

strict which means that the leftmost column spanned by the float is regarded as the column in which the float was placed.

flexible which means that the rightmost column spanned by the float is regarded as the column in which the float was placed.

These settings are only relevant if the main float/call-out relations are set to `column` or `after`.

Float and footnote relations It is possible to direct the algorithm to check on each column if there

[6] This is the LaTeX 2_ε default.

are footnotes, and if so to prevent it from placing floats in the bottom area. In theory it might be possible that a forbidden constellation might resolve itself once the algorithm has added further floats, e.g., it could be the case that by adding additional floats the offending footnote gets moved to a different column. However, checking for this would mean potentially large backtracking so the algorithm uses a conservative approach and simply considers a trial as failed if footnotes and bottom areas collide.

It is planned to allow a designer the choice of specifying where the footnotes should be placed in relation to any bottom floats (if the combination is allowed). Right now this is not implemented and column footnotes will always appear below the text column, i.e., above any bottom floats.

Area statuses For each area the algorithm keeps track of whether or not it is closed for individual float types, e.g., is not accepting any more floats of type "figure" or closed for all types. The status of an area can change due to floats being placed into other areas (this might, for example, close earlier areas, or areas that overlap) or it can change due to the fact that the area became too full in some way (e.g., a size constraint or a number of floats constraint).

Some of these constraints can be "relaxed" in certain situations, e.g., if the algorithm is directed to flush out remaining floats prior to a certain point in the galley it will drop constraints related to number of floats per area or size restrictions. However, if an area was closed due to a different float being placed into some other area, this area will stay closed in all circumstances to ensure proper sequential placement of floats and to ensure that overlapping areas that are forbidden as explained in section "Float areas" on page 280 will not receive floats at the same time.

Area constraints The algorithm offers several possibilities for the designer to specify how and under what circumstances a float is allowed to be added to a certain area on the page.

As explained above all areas on a page are tried in a specific order. This order can be specified and changed for specific parts of the document. Areas that are closed for the current type will be bypassed as well as areas which do not span the right number of columns to fit the horizontal size of the float. If these initial tests succeed the float may still fail to be placed into a certain area if it doesn't fulfill the following set of constraints:

- There is an upper limit on the total number of floats that can be placed on an individual page.
- Each area has an upper limit of floats that can go into it.
- After placing the float the remaining space in the text column must be larger than a specified value.

All such constraints are customizable.

Additional constraints will probably be implemented once there has been some experience of what controls are actually needed to allow the specification for a reasonable number of layouts.

For example, LaTeX 2ε allows the designer to restrict the maximum size of an area, but should one provide this or should there be a constraint on the size of all stacked areas? Or should there be both?

To "Here" or not to "Here" LaTeX 2ε allows the user to control the placement of an individual float by specifying one or more areas into which the float would be allowed to move using single letters. As a special notation an `h` would denote a so-called "here" float. Its advertised semantics is to try placing the float "at the position in the text where the environment appears" [1, p. 197]. If this is not feasible LaTeX 2ε would try the remaining allowed possibilities on the next page, thus a float with an `ht` specification would either appear within the text or at the top of the next or a later page.[7]

In many cases people however prefer a "here" which always means "here". The latter form is implemented in some add-on packages for LaTeX 2ε, however usually at the cost of allowing floats to appear out of order.

The new model supports only the absolute "here" form for floats; however, correct ordering of floats in the output is guaranteed (if the tag generating the here float issues flushing of floats for the current type). If there is not enough space to place the float in a column, the float plus the preceding text line[8] is moved to the next column/page.

Grid layout To produce layouts with elements placed on an underlying grid (typically with grid

[7] In two-column mode this can in fact result in a placement on the top of the second column even though the call-out position finally falls into the middle of that column.

[8] More precisely the column is broken at the last breakpoint preceding the current position which is normally one line above but could be more (or less).

points vertically separated by `\baselineskip`) the algorithm assumes that certain parts of the text column, e.g., normal text, will automatically align on the grid as long as the first line is positioned on the grid. A further assumption is that such parts of the column do not contain stretchable amounts of vertical glue so that they are not subject to stretching or shrinking if the material is adjusted to fit a given size.

Given these assumptions, the algorithm proceeds by ensuring that the space taken up by floats (including their separating white spaces) is always of a size such that the remaining space for the text part of the columns allows for an integral number of grid lines. This is achieved by stretching or shrinking the space separating the areas from the text appropriately while building the page as explained in section "Float placement concepts" on page 281.

Within the text column there are typically a number of "display objects" such as headings, equations, quotations, lists, etc., which should not be aligned on the grid. Instead, typically the text before and after is supposed to lie on the grid.[9] This is supported by allowing to mark lines of text (or more generally points in the galley) to "snap to the nearest grid point". One can think of the implementation working by taking the column material up to the marked line and putting it into a vertical box of the size of the nearest possible grid point. By this approach stretchable glue around such a display object will allow the text line that should snap to the grid to move into the correct position. This box is then given back to the page builder to assemble more material for the column. In this way the preceding part of the column becomes rigid; thus a later request for snapping to the grid will only stretch or shrink material further down the column.

A prototype implementation that makes most standard LaTeX objects, like headings, displays, etc., support grid design is available with the package `xo-grid`. It is used for typesetting this document.

User control

Column and page breaks Breaking of columns and pages can be controlled from the source document by placing special tags into it. The `\columnbreak` command ends the current column after the current line (if used in horizontal mode). Similarly the `\pagebreak` command ends the current page.[10]

Manual float flushing The flush float functionality is available within the source document via the command `\flushfloats`. This command takes two optional arguments which, if present, denote the float type to flush (by default all) and the "fuzziness" of the flush (by default `strict`). Other allowed values for the fuzziness are `column`, `page`, or `spread`. If a type is specified for flushing, effectively all types with the same float sequence class are flushed to preserve the ordering.

Specifying preferred areas At the time of writing, the document source interface for specifying the group of areas into which a float is allowed to move is not yet decided. One could envision keeping the original LaTeX interface to float environments with optional argument. In that case something like `[t]` could be internally interpreted as "any top area that exists" and translated into a list such as `t12 t11 t21`. But other interfaces are conceivable as well.

Manually position all floats Any algorithm that automatically places all floats may fail to produce adequate results in some situations. In LaTeX 2_ε the user was offered only the optional arguments of the float environments and by this method and by moving floats slightly in the source document one was finally able to change the formatting as needed.

This was a time-consuming and error-prone manual task and any slight change in the source document text was likely to result in making this work obsolete.

To improve on this situation the new algorithm can be directed to write out a file containing all of its float[11] selections (an example is shown in table 2 on the facing page). By simple drag and drop the user can produce alterations to this selection. If such a modified file is stored as `\jobname.fpc` then the algorithm will use these selections without attempting to apply any of its internal rules. Thus the formatting

[9] In some cases, depending on the design, parts of the structure might be supposed to aligned as well.

[10] At the moment these commands force a break; there is no possibility, as in LaTeX 2_ε, only to suggest that the current point is a good or bad break.

[11] Floats in this context mean "traditional" floats, not footnotes or marginpars.

will happen exactly as specified.[12]

Beside moving floats between float areas it will be possible to move floats in and out of the special area called `hhh` which represents a list of all "here" floats on the page. If a float is moved into the "here" area it means that it will be positioned as a here float at the point of its call-out.

As an extension to this method we are experimenting with restricting the manual control only to parts of the document, e.g., allowing the user to manually fix a single chapter but have the algorithm determine the remainder. We also plan to integrate column length control in this way, so that it becomes easily possible to run a page or double-spread long or short by specifying this externally rather than via tags in the source document.

Tracing the algorithm's behavior In contrast to the LaTeX 2ε output routine, which is a black box as far as the user is concerned, the new algorithm tries hard to make its decision process comprehensible. Table 3 shows a sample output produced by it. It shows for each float which areas have been tried, why they were rejected, etc. There is also an option which produces about 1000 times as much information but the latter is probably useful only for debugging the system in case there are errors in the code.

[12] If the floats are stored within the source document at the point of their call-outs, the algorithm will be able to position a float only if it has already encountered the float in the source document. This means that one can move a float arbitrarily forward but only to a limited extent before its call-out position. If the floats are stored externally to the source document this restriction does not apply.

Manually aligning text in grid layout If the algorithm produces grid layout it automatically aligns certain text lines on the underlying grid. For manual control this functionality is also provided with the command `\TextAlignGrid` which will align the current text line on the grid. By issuing a `\IgnoreAlignToGrid` command grid alignment will be temporarily disabled, while `\ObeyAlignToGrid` will reestablish automatic grid processing.

Layout Specification

In the class file the designer is given control over the algorithm's behavior in all the aspects described above (and several more).

The layout specifications are done through the new template and instance concept, see [2]. Addi-

Table 2: An example `fpl` file

```
Page: 1 (1)
    Area: t13
        Float: 4 (figure 4) []
    Area: b21
        Float: 2 (figure 2) [mylab:fig1]
    Area: t31
        Float: 3 (figure 3) [mylab:fig2]

    Area: hhh
        Float: 11 (table 1) []

Page: 2 (2)
    Area: t13
        Float: 8 (figure 8) []
    Area: t22
        Float: 5 (figure 5) []
    Area: b11
        Float: 6 (figure 6) [mylab:fig3]
    Area: b31
        Float: 7 (figure 7) [mylab:fig4]
```

Table 3: Progress output of the algorithm

```
======================================================================
 STATS: floats waiting = 2 on page 13
======================================================================
Float: \bx@E   {5}  {table} (floats)  {5}   {Statistics from the algorithm}
 area trial: b12  -> failed: span count b12 /= 1
 area trial: b11  -> accepted
Float: \bx@F   {6}  {table} (floats)  {6}   {Running times of the algorithm}
 area trial: b12  -> failed: span count b12 /= 1
 area trial: b11  -> failed: b11 float num reached (1)
 area trial: b21  -> failed: area below flush point (2=2, b21)
 -> failed: --> retry with relaxed conditions

 area trial: b12  -> failed: span count b12 /= 1
 area trial: b11  -> accepted
 STATS: trials = 7
```

tional information such as experimental code, further documentation, etc., can be found on the LaTeX project web site at:

`http://www.latex-project.org`

In contrast to the algorithm itself, which in its basic functionality now seems to be stable and reliable, the design interface is far more experimental. Thus the example declarations given below represent only the current state of thought (or of implementation) and are likely to be modified at any moment.

Float type declarations Float types are declared using the command `\DeclareFloatType` which takes two arguments: the name of the type which is declared and in the second argument a list of key/value pairs which describe the properties of the float type, e.g.,

```
\DeclareFloatType{figure}
  {
   sequence-class-id  = floats,
   toc-extension      = lof,
   caption-text       = \figurename,
   numbered-boolean   = true,
   numbered-id        = figure,
   numbered-within-id = section,
   numbered-action    =
        \thesection.\arabic{figure},
   body-decls         = ,
  }
```

The `sequence-class-id` key defines to which float sequence class the type belongs to. If it is absent a sequence class with the same name as the type is assumed. The sequence class will be automatically initialized if not referenced before.

The `toc-extension` key defines the extension to be used to write the caption to when generating "List of floats" listings. By using the same extension with different types it is possible to generate combined listings, such as "List of tables and figures".

The `caption-text` key defines the fixed text to be used as part of the caption text together with the float number if present, e.g., `Figure`. This information is passed to the caption formatting template so the actual formatting is defined there.

The `numbered-boolean` defines whether or not floats of this type are numbered.

The `numbered-id` key defines the name of the counter to use when numbering floats. If absent a counter with the same name as the type is assumed. By using the same counter with different types it is possible to use a single numbering scheme—in that case the `sequence-class-id` for these types should probably be identical as well to avoid strange numbering sequences within the document.

The `numbered-within-id` key defines the name of the "within" counter, i.e., the counter which if stepped resets the numbering. If the value is empty or not set the float type is numbered in a single sequence throughout the document.

The `numbered-action` key defines the representation of the float number, as used in the caption and by the `\ref`, `\label` mechanism. The default is `\arabic{`⟨*counter*⟩`}`.

The `body-decls` key can hold formatting instructions that should apply to the float body. They can assume a normalized formatting environment already set up by the algorithm.

The declaration of a new float type automatically defines the necessary user document environments.

Float area declarations Any float area that is going to be used at some stage by the algorithm needs to be declared beforehand. This is done through the `\DeclareFloatArea` command which takes two arguments: the name of the area (which has to follow the conventions explained in section "Float areas" on page 280) and a list of key/value pairs describing the characteristics of the area.

```
\DeclareFloatArea{t22}
   {
     class-close-list = {t11,b11},
     all-close-list   = {t12,t32},
     max-float-num    = 2,
   }
```

As of today an area is characterized through the maximum number of floats it is allowed to receive (`max-float-num`) and through two lists which tell the algorithm which other areas are affected by adding a float to the current area. The list `class-close-list` enumerates all areas which are not allowed to receive additional floats of the same sequence class as the float that has been placed into the current area, while the list `all-close-list` contains the information about all areas that are to be completely closed the moment a float is received in the current area.

The `class-close-list` key is primarily intended to specify a partial order on the areas to ensure that floats are not getting out of sequence in the

output. For example, the above declaration says: if a float is placed into area `t22`, i.e., a top area starting at column two and spanning two columns, then the single column areas `t11` and `b11` (i.e., those of the first column) are closed for floats of the same class. However, assuming this example is part of a declaration for a four column layout which could have areas like `t14` or `t13`, there is nothing said about closing those areas. Thus in this particular layout a float spanning three or four columns would still be allowed to go on top.

On the other hand the `all-close-list` key is available to ensure more visual constraints, e.g, "if `t12` gets filled we don't want to have `b12` filled as well, we only want `b22` in this case." In addition it is needed to implement the restriction about overlapping float areas as described in section "Float areas" on page 280, e.g., in the example declaration `t12` and `t32` are closed since they partly overlap with `t22`.[13]

Footnote formatting declarations The formatting of footnotes is specified by declaring instance(s) of type `footnotesetup`. At the moment three templates are available though they should be considered only as prototypes: the template `std` produces conventional footnotes below each column, the template `ftnright` collects all footnotes and typesets them in the rightmost column, and the `margin` template collects and typesets them in the right outer margin.

The keys of the above templates provide only a rudimentary flexibility (to say it positively); in a production version all of them would need a large number of extensions. As an example,

```
\DeclareInstance{footnotesetup}
  {mainmatter}{std}
  {
   text-sep    = 14pt plus 3pt,
   max-height  = 8in,
  }
```

would declare the named instance `mainmatter` that provides footnotes below columns with a separation of `14pt+` and a maximum height for footnotes per column being `8in`.

Instances like this can then be used in the declaration for a particular page layout as explained below.

[13] As mentioned before, this restriction might be lifted in a later version of the algorithm; as long as it is required one could alternatively add those areas behind the scenes to avoid runtime problems.

Alternatively one could use unnamed instances there using the `\UseTemplate` method.

Page setup declarations At the heart of the layout declaration are instances of the type `pagesetup2`.[14]

An example setup showing all currently available keys is given in table 4 on the following page.

Column specification The first four keys (`column-num`, `column-width`, `column-height`, and `column-sep`) describe the column structure of the page layout being defined, i.e., in this case a two-column layout.

Float constraint specification The following four keys define the standard constraints for the algorithm when placing floats: `max-float-num` is the maximum number of floats that can go on a normal page; `float-callout-constraint` defines what kind of relations between float and call-out are allowed (possible values are explained on page 282); `float-callout-span-constraint` handles the interpretation of spanning floats and is explained on page 282; and `bottom-float-footnote-constraint` defines whether or not bottom floats are allowed in case of footnotes.

The last three constraints are replaced by `flush-float-callout-constraint`, `flush-float-callout-span-constraint`, and `flush-bottom-float-footnote-constraint` in case flushing can't be done without relaxing the conditions (`max-float-num` is disregarded in that case automatically).

Float area specification The key `area-list` defines all float areas that are allowed in this page layout as well as defining the order in which the areas are tried when placing floats. The keys `defer-class-close-list` and `defer-all-close-list` define the "closing actions" for the special area which receives the floats that could not be placed. E.g., if a float of a certain class can't be placed then all areas listed in `defer-class-close-list` will be closed for this class of floats. In other words the two keys are comparable to the ones available for area declarations.

Thus these keys together with the keys from the area declarations are most important to guarantee a sensible order of floats on the formatted page.

In an earlier implementation of the algorithm a simpler scheme was used: there was a single area list which was shortened whenever a float couldn't be

[14] The number 2 has historical reasons and will vanish at some point in the future.

placed into it thereby confining the remaining floats to this restricted selection. This works fine as long as there are mainly single column floats since in this case the area can be reasonably ordered into a single sequence. However the moment spanning floats are supported the situation gets less straightforward. Is it allowed to place a later float into `t12` if there is already a float in the area `t11`?

It is quite likely that the current controls will turn out to be too crude. This will be seen once a suitable number of layouts have been produced under this scheme (or couldn't be produced because they turned out to be unspecifiable).

There needs to be space between floats in an area and areas need to be separated from each other, as well as from the column text. For this we have the following keys: `float-float-sep` is the separation between two floats in an area, `float-area-sep` is the separation between two vertically adjacent areas, and `float-text-sep` finally is the separation between a float area and the column text.[15] The separation between inline floats and surrounding text is given by `float-inline-sep`.

Grid specification To produce a grid based design the `grid-point-sep` needs to be given a positive dimension. This defines the distance between grid points on which the algorithm aligns column text, inline floats, etc.[16]

To align column text at a grid point the algorithm will extend the `float-text-sep` space. Alternatively, if the nearest grid point can be reached by shrinking that space (assuming its specification contains a `minus` component) the algorithm will use that grid point instead. In a similar fashion the space around an inline float will be determined by the value of `float-inline-sep`.

Footnote, etc., specification Finally the key `footnote-setup` receives an instance of a `footnotesetup` template, thereby defining how footnotes are handled and presented.

What is clearly missing here is handling of other page elements such as running headers and footers,

[15] A possible extension would be to allow ornamental material in place of white space.

[16] Setting this parameter is not sufficient: to make grid setting possible several other parameters need to be set to suitable values as well, e.g., the distance between baselines should be compatible and the column height needs to be a multiple of this value.

Table 4: Example declaration for the `pagesetup2` template showing all currently available keys

```
  \DeclareInstance{pagesetup2}{mainmatter}{std}
    {
% column specification
     column-num                              = 2,
     column-width                            = 220pt,
     column-height                           = 610pt,
     column-sep                              = 20pt,
% float constraint specification
     max-float-num                           = 3,
     float-callout-constraint                = after,
     float-callout-span-constraint           = strict,
     bottom-float-footnote-constraint        = forbidden,
     flush-float-callout-constraint          = page,
     flush-float-callout-span-constraint     = flexible,
     flush-bottom-float-footnote-constraint = none,
% area specification
     area-list                               = {t12,t11,b11,b12,t21,b21},
     defer-class-close-list                  = {t12,t11,b11,b12,t21,b21},
     defer-all-close-list                    = ,
     float-float-sep                         = 15pt,
     float-text-sep                          = 30pt minus 8pt,
     float-area-sep                          = 15pt,
     float-inline-sep                        = 6pt minus 2pt,
% grid specification
     grid-point-sep                          = 12pt,
% footnote etc specification
     footnote-setup                          = mainmatter,
    }
```

the folio, etc. This will be added soon.

Float formatting declarations For the attachment of captions to floats there exists a prototype interface using templates of the type `buildfloat`. At the time of writing, available templates are `centeredbelow`, `centeredabove`, and `bottomright`, which center the caption below or above the float body or place it to the right of it, aligned with the bottom of the float body. All of them would need to be generalized for a production system to become more flexible.

When trial-formatting a float the algorithm checks for the existence of a number of `buildfloat` instances and uses the first one that exists to build the float. More precisely it first checks if an instance with the name ⟨*area*⟩-⟨*type*⟩ exists, then it looks for ⟨*area*⟩, then for ⟨*type*⟩, and finally, if none of them exists, for an instance with the name `default`. So at least the latter instance has to be declared by the class.

```
\DeclareInstance{buildfloat}{default}
   {centeredbelow}{}
\DeclareInstance{buildfloat}{table}
   {centeredabove}{}
\DeclareInstance{buildfloat}{t31}
   {bottomright}{}
\DeclareInstance{buildfloat}{t22}
   {bottomright}{}
```

The example declaration above defines the placement of captions above tables and below for all other types, with the exception of the areas `t31` and `t22` where the captions are set to the side.

Performance of the algorithm

To test the performance of the algorithm we prepared a somewhat ridiculous test file containing three types of floats ("figures", "tables", and "algorithms") with a total number of 47 floats. The chosen layout had 3 columns and 11 potential float areas. Figure captions have been placed below the float while with tables and algorithms the caption was placed on top. The exception was the top areas adjacent to the outer margin: floats placed there got their captions placed to the right and partly into the margin. Footnotes were collected for all columns and placed in the outer margin.

Floats had to strictly follow their call-out and a maximum of ten floats was allowed per page, i.e., roughly three per column.

Since the document contained many floats early on (24 on page one) and the first of these was especially constructed to be not placeable the first time around, the algorithm had to work hard to place all the dangling floats. Table 5 shows some statistics as produced by the algorithm on the number of trials necessary (the highest number was 397 for 37 floats; by comparison, equation (1) on page 279 would give 22595200368 which would probably take a bit longer to evaluate). Note that on the third page the algorithm was able to produce a float page; on all other pages the float page trial was unsuccessful.

Table 6 on the following page shows the running times needed to produce the final document of 13 pages when the algorithm is used with different tracing settings. The test machines were a Pentium III 650 machine and an older laptop with a 486 processor. In both cases TEX was run straight from a TEX Live 4 CD.

These times show that the algorithm has an acceptable time performance since even on a 486 the average time to produce a page is roughly 2 seconds.

Outlook

While the current algorithm performs well there are

Table 5: Statistics from the algorithm

```
STATS: floats waiting = 24 on page 1
STATS: trials = 286
STATS: floats waiting = 19 on page 2 (float page)
STATS: trials = 159
STATS: floats waiting = 37 on page 2
STATS: trials = 397
STATS: floats waiting = 19 on page 3 (float page)
STATS: trials = 166
STATS: floats waiting = 7 on page 4 (float page)
STATS: trials = 41
STATS: floats waiting = 20 on page 4
STATS: trials = 204
STATS: floats waiting = 5 on page 5 (float page)
STATS: trials = 27
STATS: floats waiting = 12 on page 5
STATS: trials = 108
STATS: floats waiting = 0 on page 6 (float page)
STATS: trials = 0
STATS: floats waiting = 6 on page 6
STATS: trials = 57
         ...
STATS: floats waiting = 6 on page 12 (float page)
STATS: trials = 26
STATS: floats waiting = 6 on page 12
STATS: trials = 37
STATS: floats waiting = 0 on page 13
STATS: trials = 0
```

several areas in which its functionality could and probably should be extended. The most important points are given in the following list.

- Balancing of partial pages, comparable to the way the `multicol` package works, should be implemented to allow for layouts where, for example, a heading should span across all columns.
- We intend to provide more control over the marginal areas, allowing for marginal floats as well as other objects in the margin, properly interacting with each other.
- Without much effort the algorithm could be extended to properly support double-spreads so this should be added some time soon.
- Once the algorithm has decided which floats to place onto a page one could add a post-processing step in which the placement could be reconsidered according to different rules. For example, if the call-out relation is `page` then floats will tend to be placed in the left-hand columns. This is fine as long as there are many floats to process but on a page with only a few floats one might want to redistribute them differently once it is clear which floats could go onto the page.
- Since it is known beforehand how many floats are actively waiting to be placed, one could use a different algorithm that tries all possible combinations as long as there are only a limited number of floats to be placed. The boundary at which the algorithm changes behavior could be made customizable so that people with faster machines (or more patience) could have the search for optimum running for as many floats as they like.

Table 6: Running times of the algorithm

	PIII (650MHz)	486DX4 (75MHz)
	no tracing	
real	0m1.533s	0m27.633s
user	0m1.460s	0m26.940s
sys	0m0.050s	0m0.690s
	progress information	
real	0m3.116s	0m36.885s
user	0m1.740s	0m34.470s
sys	0m0.080s	0m2.420s
	full tracing	
real	0m7.833s	1m22.480s
user	0m2.720s	1m7.890s
sys	0m0.280s	0m12.360s

References

[1] Leslie Lamport. *LaTeX: A Document Preparation System*. Addison-Wesley, Reading, Massachusetts, second edition, 1994.

[2] Frank Mittelbach, David Carlisle, and Chris Rowley. New interfaces for LaTeX class design. *TUGboat*, 20(3):214–216, September 1999.

Frank Mittelbach