Effective Float Strategies

DocEng Conference 2017, Malta

Frank Mittelbach
frank.mittelbach@latex-project.org

LaTeX3 Project

September, 2017

Pagination with floats

- Last year's starting point
- Visualizing the general approach
- Adding floats to the mix
- Results
- Comparisons

John Tenniel, 1870

Last year's starting point
(text-only case — no floats)

Pagination with greedy algorithm fails for nearly 100% of the cases

Idea: use dynamic programming approach (e.g., Knuth/Plass) for pagination

Doable . . . Complexity is

- $O(n)$ for fixed spread structure
- $O(n^2)$ otherwise

But . . .

- there is not enough flexibility in a page
- Thus: most of the time optimizing runs out of options

Add enough flexibility . . . through

- spread height variations (run them long or short)
- paragraph variations (format to different heights)

Last year's starting point

(text-only case — no floats)

Pagination with greedy algorithm fails for nearly 100% of the cases

Idea: use dynamic programming approach (e.g., Knuth/Plass) for pagination

Doable … Complexity is

- $O(n)$ for fixed spread structure
- $O(n^2)$ otherwise

But …

- there is not enough flexibility in a page
- Thus: most of the time optimizing runs out of options

Add enough flexibility … through

- spread height variations (run them long or short)
- paragraph variations (format to different heights)

Last year's starting point

(text-only case — no floats)

🙁 Pagination with greedy algorithm fails for nearly 100% of the cases

🤔 Idea: use dynamic programming approach (e.g., Knuth/Plass) for pagination

Doable ... Complexity is

- $O(n)$ for fixed spread structure
- $O(n^2)$ otherwise

But ...

- there is not enough flexibility in a page
- Thus: most of the time optimizing runs out of options

Add enough flexibility ... through

- spread height variations (run them long or short)
- paragraph variations (format to different heights)

Last year's starting point

(text-only case — no floats)

Pagination with greedy algorithm fails for nearly 100% of the cases

Idea: use dynamic programming approach (e.g., Knuth/Plass) for pagination

Doable ... Complexity is

- $O(n)$ for fixed spread structure
- $O(n^2)$ otherwise

But ...

- there is not enough flexibility in a page
- Thus: most of the time optimizing runs out of options

Add enough flexibility ... through

- spread height variations (run them long or short)
- paragraph variations (format to different heights)

Last year's starting point

(text-only case — no floats)

Pagination with greedy algorithm fails for nearly 100% of the cases

Idea: use dynamic programming approach (e.g., Knuth/Plass) for pagination

Doable … Complexity is

- $O(n)$ for fixed spread structure
- $O(n^2)$ otherwise

But …

- there is not enough flexibility in a page
- Thus: most of the time optimizing runs out of options

Add enough flexibility … through

- spread height variations (run them long or short)
- paragraph variations (format to different heights)

Last year's starting point
(text-only case — no floats)

Pagination with greedy algorithm fails for nearly 100% of the cases

Idea: use dynamic programming approach (e.g., Knuth/Plass) for pagination

Doable … Complexity is

- $O(n)$ for fixed spread structure
- $O(n^2)$ otherwise

But …

- there is not enough flexibility in a page
- Thus: most of the time optimizing runs out of options

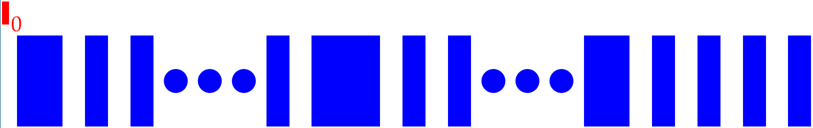
Add enough flexibility … through

- spread height variations (run them long or short)
- paragraph variations (format to different heights)

The battlefield

A visualization of the algorithms

John Tenniel, 1870

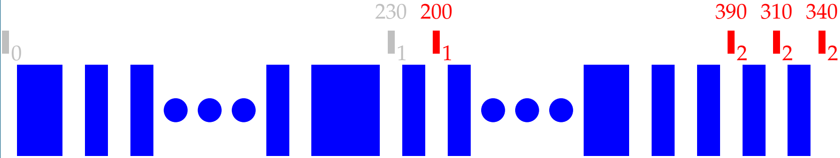


The galley in need of pagination

- Blue bars represent (blocks of) lines
- Breaks happen only between blocks
- Above we mark candidate breaks (active nodes) in red

Visualization of the algorithm

The Basics

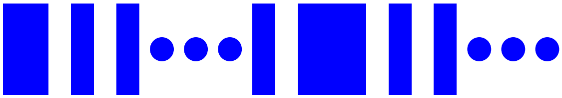


Active nodes

- Sliding window into the document
- Add new node when breakpoint can end a page
- Only the best solution (accumulated costs) is used
- Deactivate when too far from current point

Visualization of the algorithm

The basics

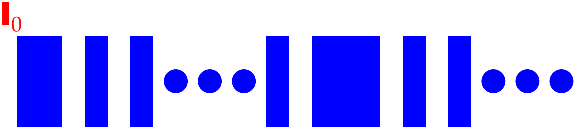


Step

- Generate a galley from source material . . .

Visualization of the algorithm

The basics

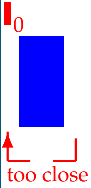


Step

- Make active node representing document start . . .

Visualization of the algorithm

The basics

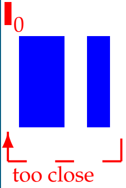


Step

- Loop through breakpoints and try to make a page …

Visualization of the algorithm

The basics

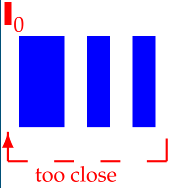


Step

- Loop through breakpoints and try to make a page . . .

Visualization of the algorithm

The basics

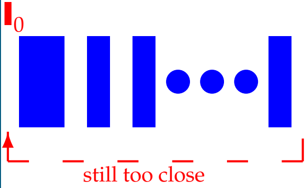


Step

- Loop through breakpoints and try to make a page . . .

Visualization of the algorithm

The basics

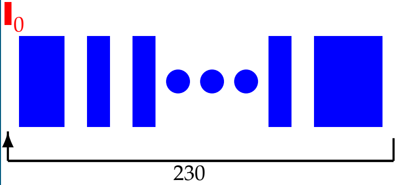


Step

- Loop through breakpoints and try to make a page ...

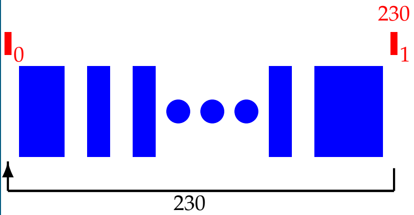
Visualization of the algorithm

The basics



Step

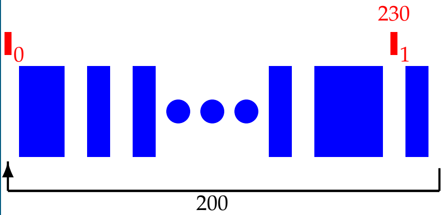
- …first success (costs 230) …



Step

- Make active node representing solution . . .

Visualization of the algorithm
The basics

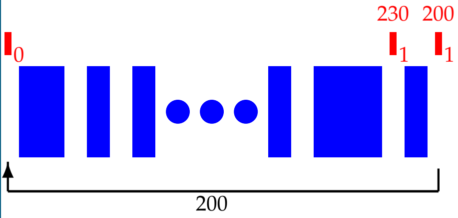


Step

- . . . next success (costs 200) . . .

Visualization of the algorithm

The basics

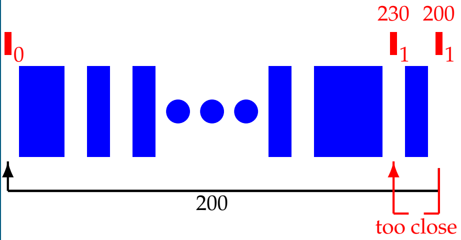


Step

- Make active node representing solution …

Visualization of the algorithm

The basics

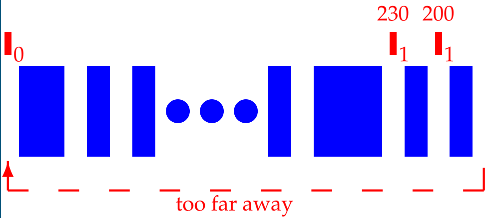


Step

- Try second active node . . . (fail) . . .

Visualization of the algorithm

The basics

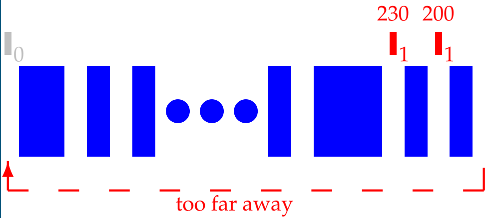


Step

- Try making page (fail) . . .

Visualization of the algorithm

The basics

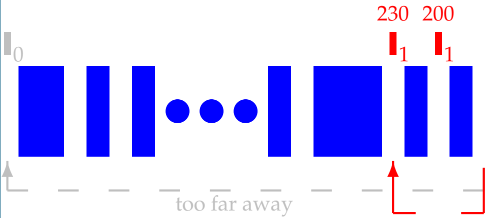


Step

- Disable active node too far away …

Visualization of the algorithm

The basics

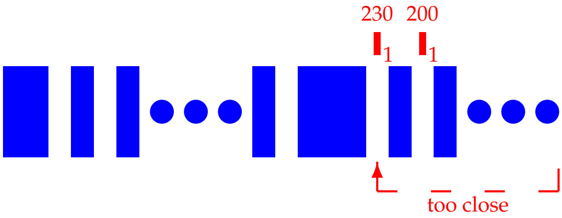


Step

- Try next active node (fail) . . .

Visualization of the algorithm

The basics

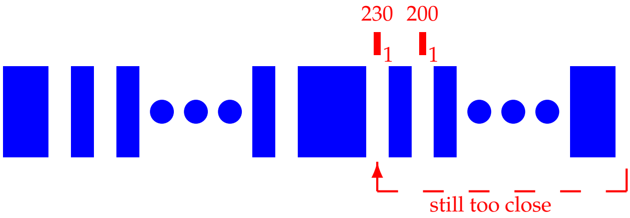


Step

- Other active nodes will fail too, so try next breaks …

Visualization of the algorithm

The basics

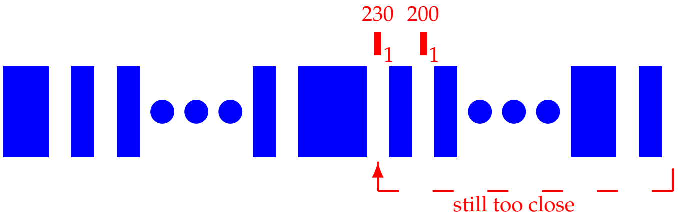


Step

- Continue trying . . .

Visualization of the algorithm

The basics

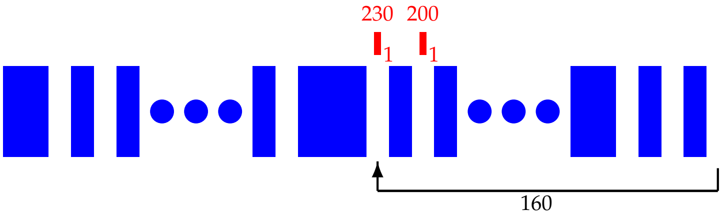


Step

- Continue trying ...

Visualization of the algorithm

The basics

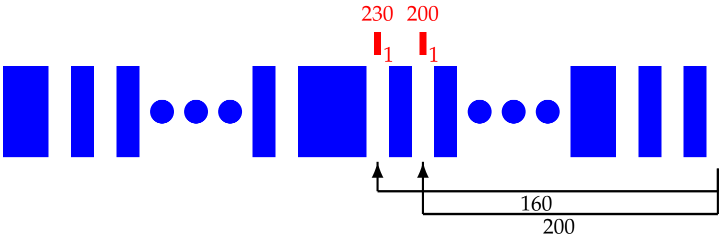


Step

- …success (costs 160) …

Visualization of the algorithm

The basics

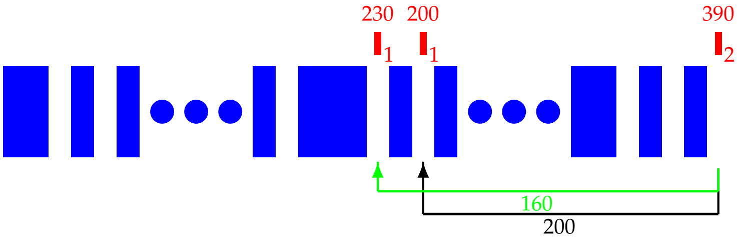


Step

- . . . alternative solution (costs 200) . . .

Visualization of the algorithm

The basics

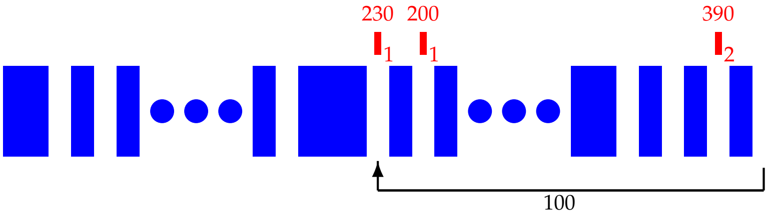


Step

- Make active node for best solution . . .

Visualization of the algorithm

The basics

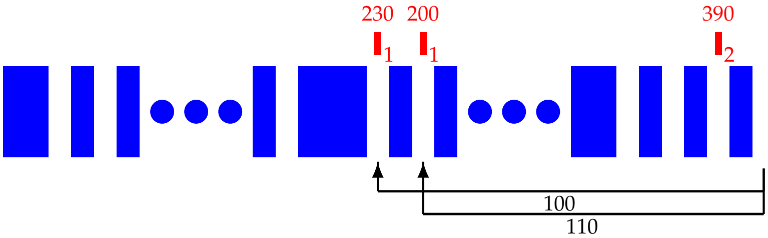


Step

- Next break with solution (costs 100) . . .

Visualization of the algorithm

The basics

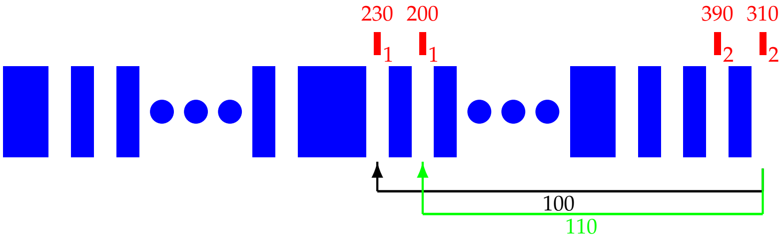


Step

- …alternative solution (cost 110) …

Visualization of the algorithm

The basics

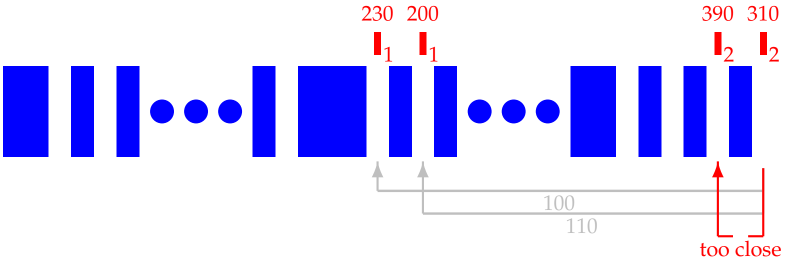


Step

- Second solution is best overall, make active node . . .

Visualization of the algorithm

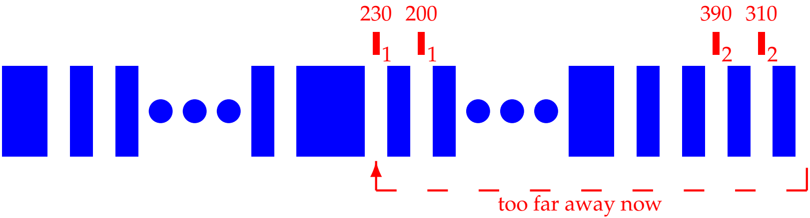
The basics



Step

- Try next active node to make third page (fail) . . .

Visualization of the algorithm
The basics

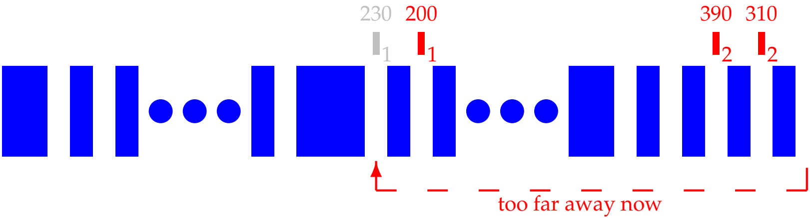


Step

- Try next break with first active node . . .

Visualization of the algorithm

The basics

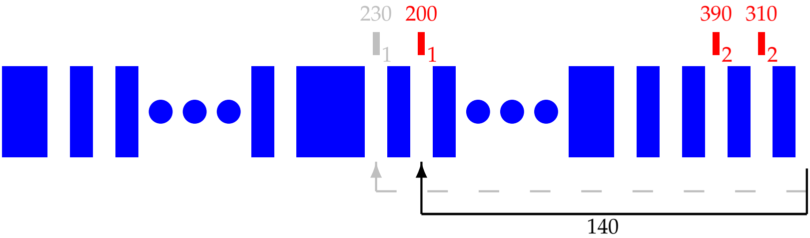


Step

- … disable active node (too far away) …

Visualization of the algorithm

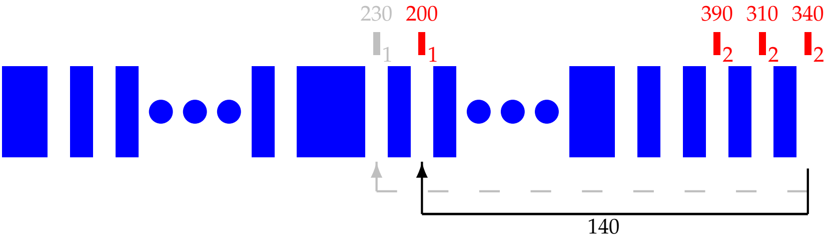
The basics



Step

- Try next active node (success, costs 140) …

Visualization of the algorithm
The basics

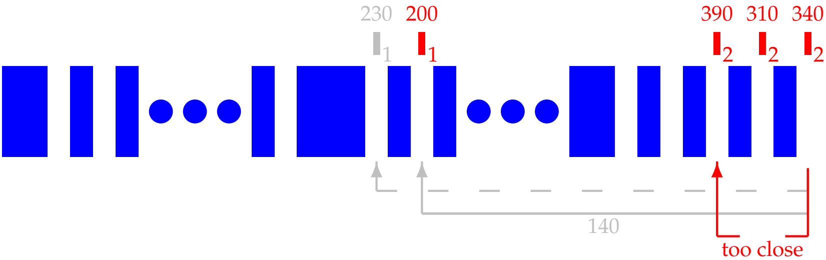


Step

- Make new active node for solution …

Visualization of the algorithm

The basics

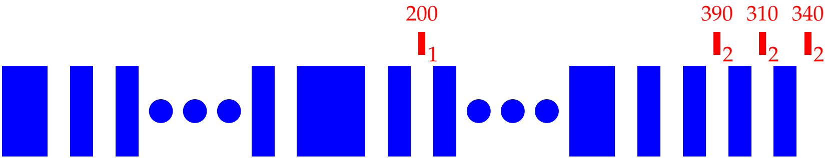


Step

- Look at active nodes ending second page (fail) . . .

Visualization of the algorithm

The basics

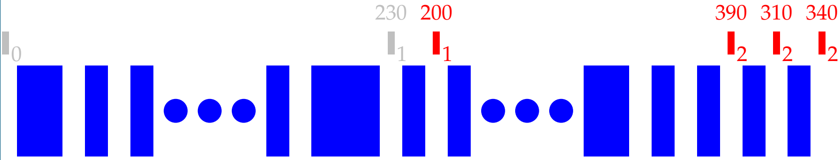


Step

- … continue with next break …

Visualization of the algorithm

Complexity

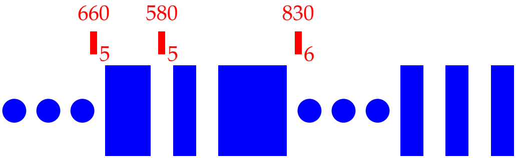


Complexity

- Loop through all n breakpoints
  - and try making pages back to each active node
- Thus the complexity is
  - $O(n \times \langle \text{average length of active list} \rangle)$

Visualization of the algorithm

Pages have identical heights

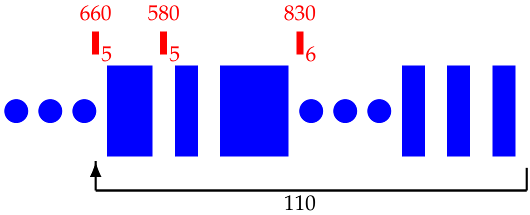


In later parts of the document ...

- active nodes for different pages may get close together

Visualization of the algorithm

Pages have identical heights

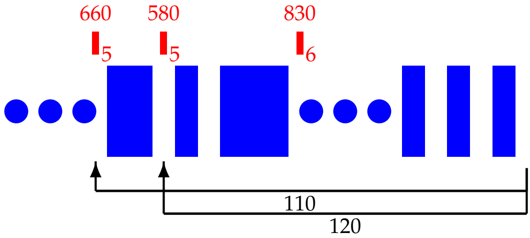


Step

- One candidate solution …

Visualization of the algorithm

Pages have identical heights

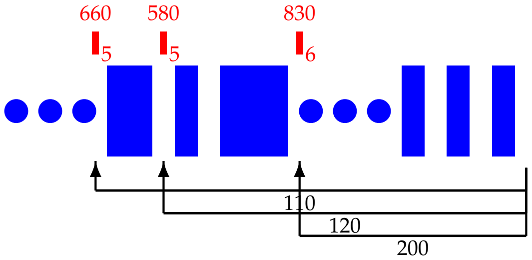


Step

- Another candidate solution …

Visualization of the algorithm

Pages have identical heights

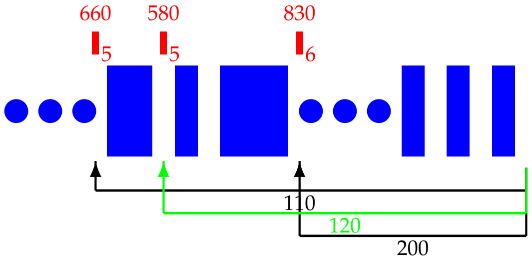


Step

- And as page height is identical this one competes too . . .

Visualization of the algorithm

Pages have identical heights

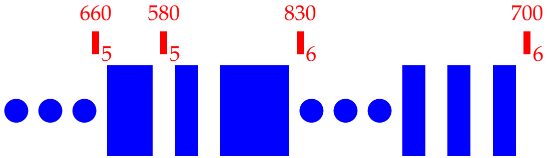


Step

- So we only need to remember the best of them . . .

Visualization of the algorithm

Pages have identical heights

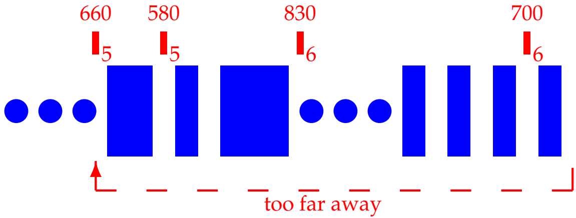


Step

- … and make one active node for it …

Visualization of the algorithm

Pages have identical heights

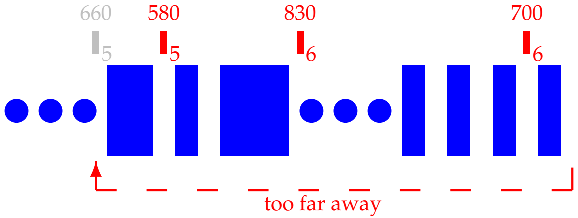


Step

- And . . .

Visualization of the algorithm

Pages have identical heights

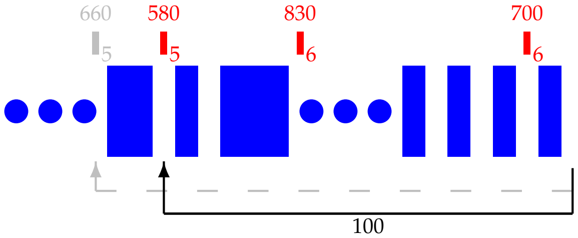


Step

- And so …

Visualization of the algorithm

Pages have identical heights

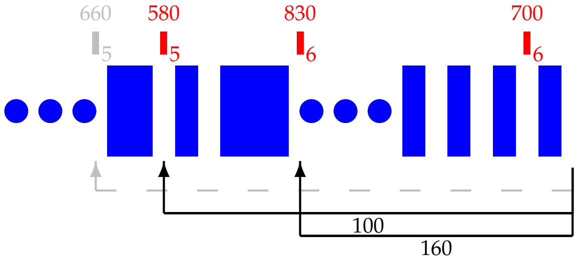


Step

- And so on …

Visualization of the algorithm

Pages have identical heights

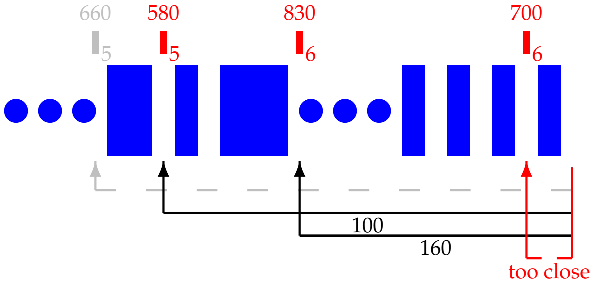


Step

- And so on with . . .

Visualization of the algorithm

Pages have identical heights

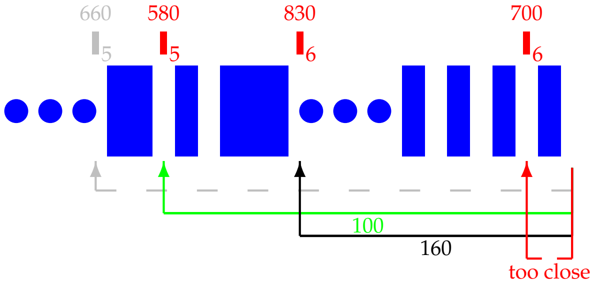


Step

- And so on with all …

Visualization of the algorithm

Pages have identical heights

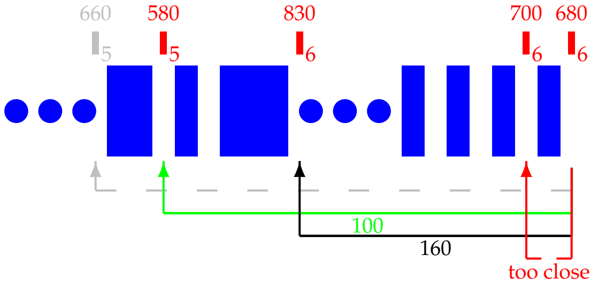


Step

- And so on with all further …

Visualization of the algorithm

Pages have identical heights

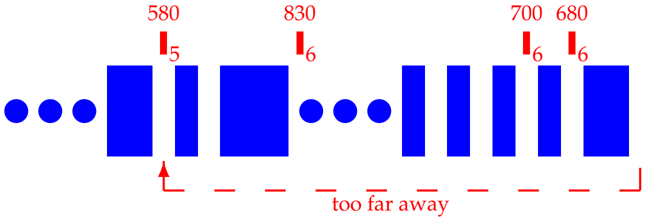


Step

- And so on with all further breaks ...

Visualization of the algorithm

Pages have identical heights

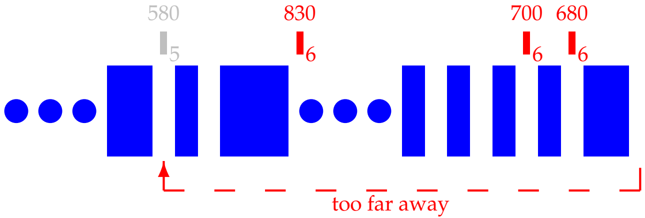


Step

- And so on with all further breaks …

Visualization of the algorithm

Pages have identical heights

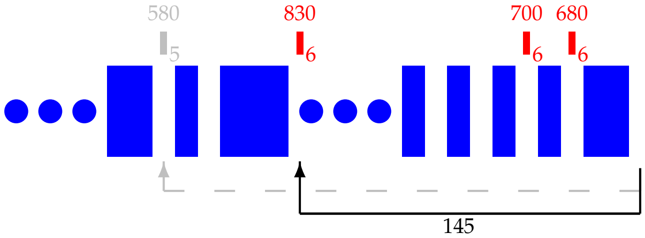


Step

- And so on with all further breaks . . .

Visualization of the algorithm

Pages have identical heights

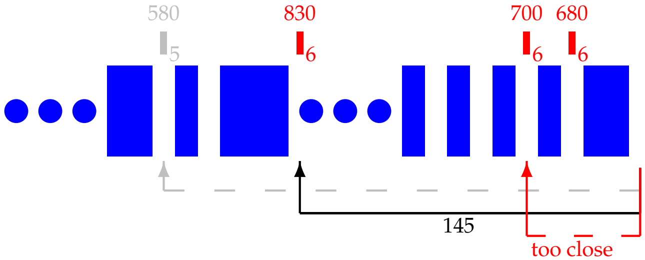


Step

- And so on with all further breaks …

Visualization of the algorithm

Pages have identical heights

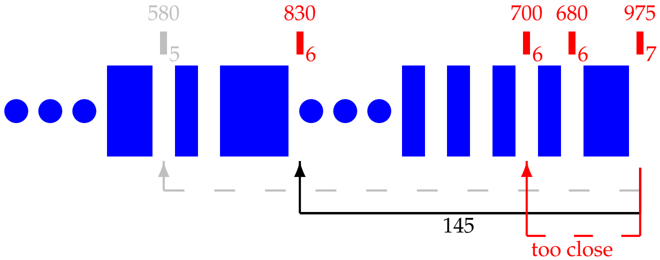


Step

- And so on with all further breaks …

Visualization of the algorithm

Pages have identical heights

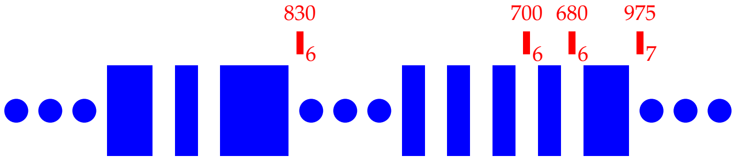


Step

- And so on with all further breaks . . .

Visualization of the algorithm

Pages have identical heights

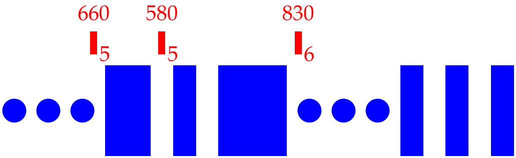


Step

- And so on with all further breaks … (total of 4)

Visualization of the algorithm

Pages have different heights

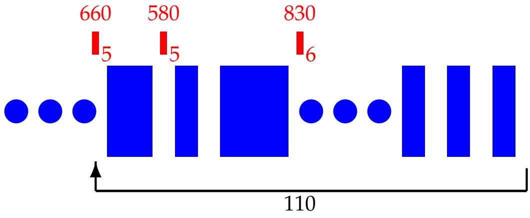


Different page heights are the more complex situation

- as we will have more active nodes to deal with ...

Visualization of the algorithm

Pages have different heights

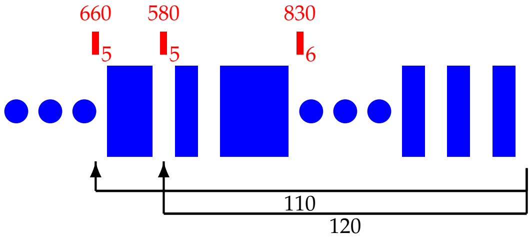


Step

- Trying to build page 6 (success) . . .

Visualization of the algorithm

Pages have different heights

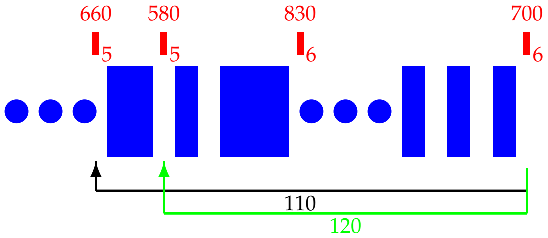


Step

- … an alternative (slightly higher costs) …

Visualization of the algorithm

Pages have different heights

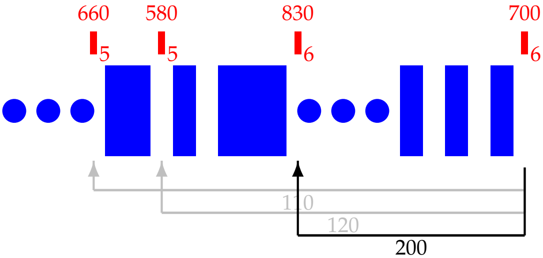


Step

- but overall better . . .

Visualization of the algorithm

Pages have different heights

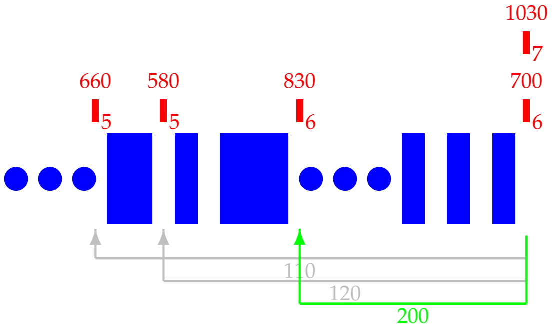


Step

- We can also make page 7 end here …

Visualization of the algorithm

Pages have different heights

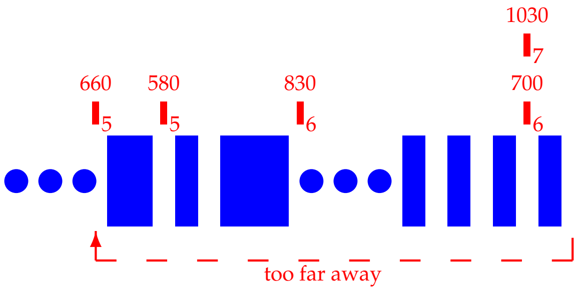


Step

- … thus we make another active node …

Visualization of the algorithm

Pages have different heights

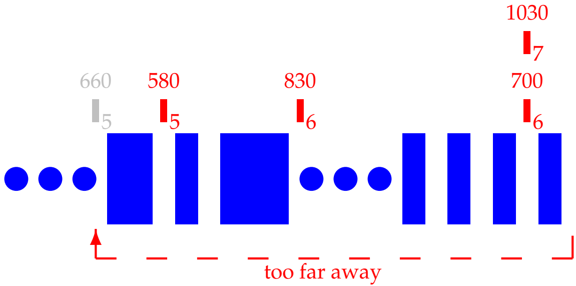


Step

- Try next break . . .

Visualization of the algorithm

Pages have different heights

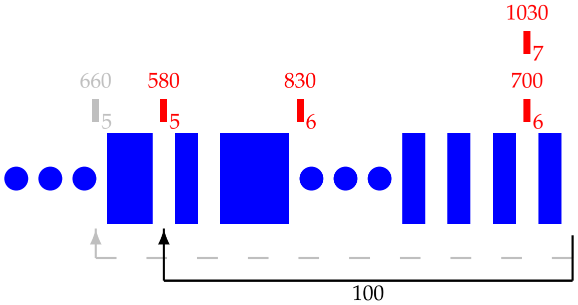


Step

- First active node no longer reachable, thus disable …

Visualization of the algorithm

Pages have different heights

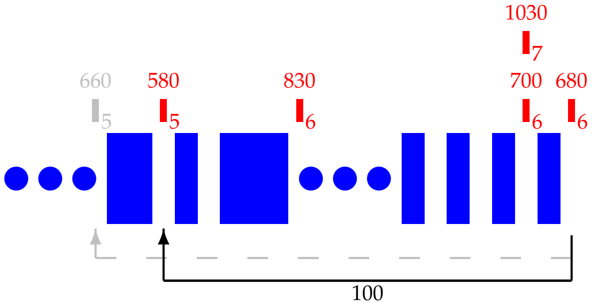


Step

- Next one is possible (costs 100) …

Visualization of the algorithm

Pages have different heights

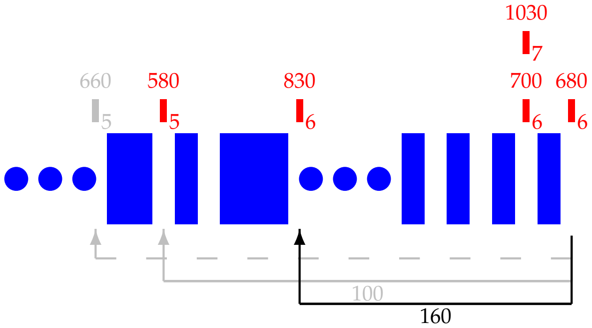


Step

- … thus another solution for page 6 …

Visualization of the algorithm

Pages have different heights

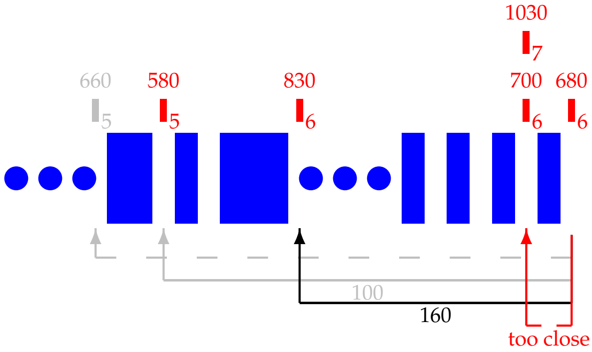


Step

- A candidate for page 7 (costs 160) ...

Visualization of the algorithm

Pages have different heights

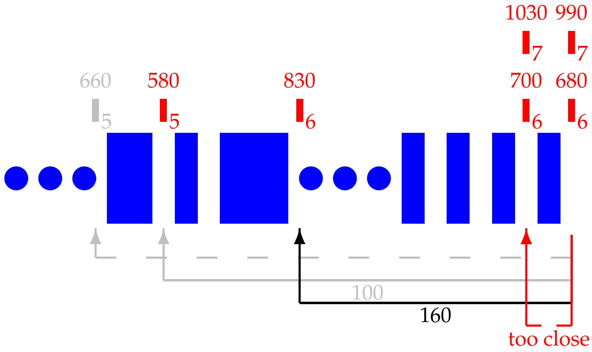


Step

- In fact the only one (as this one is too short), so …

Visualization of the algorithm

Pages have different heights

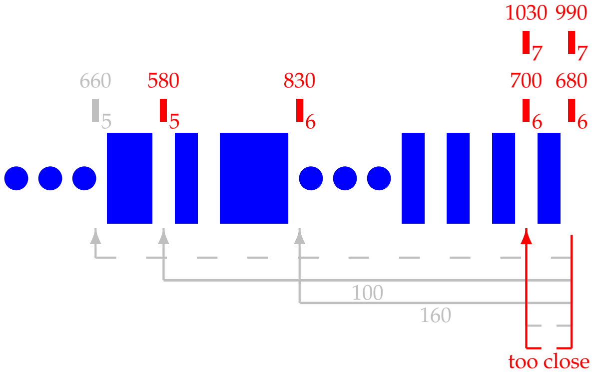


Step

- … make yet another active node …

Visualization of the algorithm

Pages have different heights

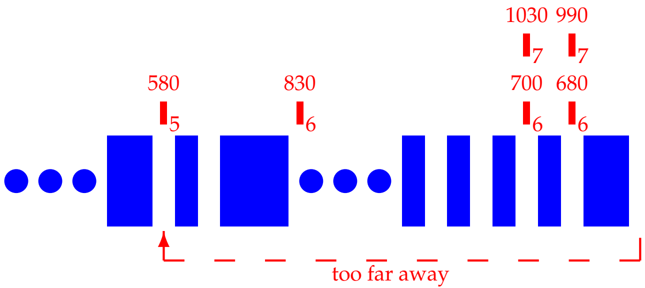


Step

- Ending page 8 here doesn't work ...

Visualization of the algorithm

Pages have different heights

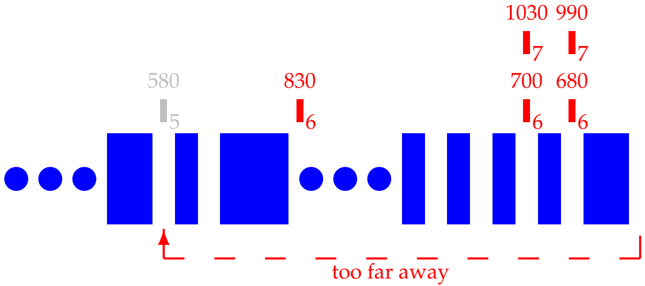


Step

- Next break is too far from first active node . . .

Visualization of the algorithm

Pages have different heights

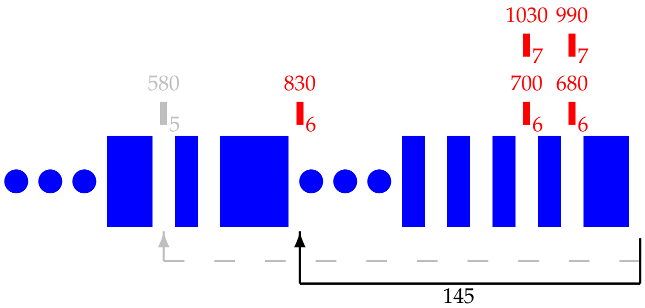


Step

- … so we disable it …

Visualization of the algorithm

Pages have different heights

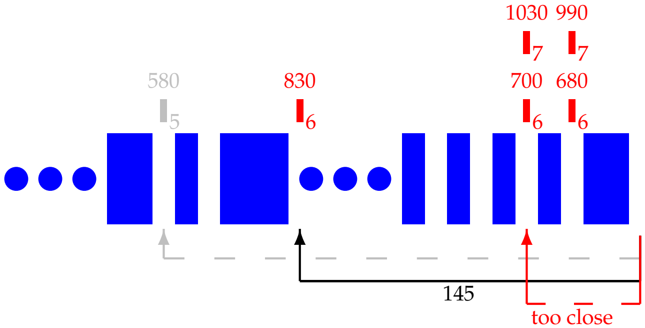


Step

- But next active node is a candidate for page 7 ...

Visualization of the algorithm

Pages have different heights

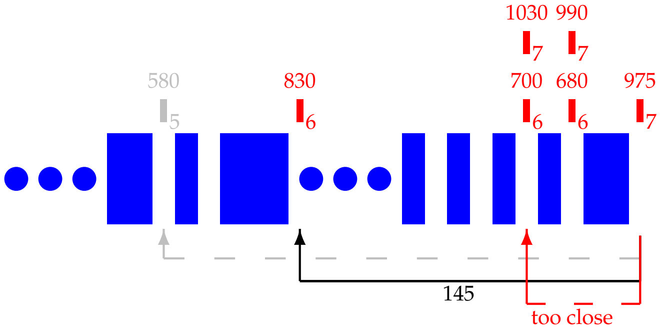


Step

- In fact the only one, so …

Visualization of the algorithm

Pages have different heights

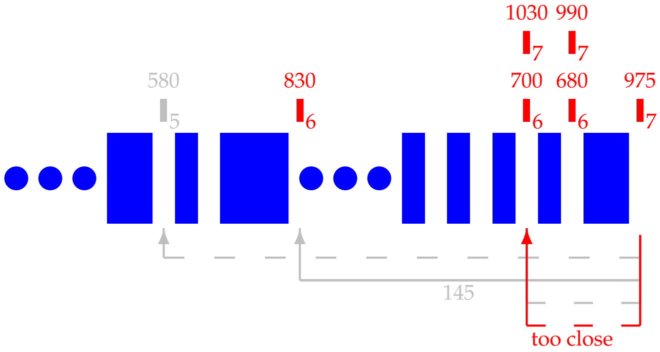


Step

- … make yet another active node …

Visualization of the algorithm

Pages have different heights

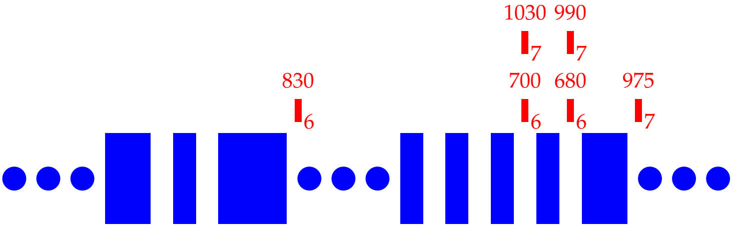


Step

- Page 8 is not possible from here, so we are done …

Visualization of the algorithm

Pages have different heights



Step

- … and so on and so forth … (total of 6 in this example)

Visualization of the algorithm

Complexity II

Pages have identical heights (after some page)

- Active list is bounded by a constant
- Thus
  - $O(n \times \langle \text{average length of active list} \rangle) = O(n)$

Pages have varying heights

- Active list can grow arbitrarily (i.e., $O(n)$)
- Thus we end up with
  - $O(n \times \langle \text{average length of active list} \rangle) = O(n^2)$

Spread and paragraph variations

- They add a factor of $O(1)$ to the length of the active list
- Thus the complexity doesn't change!

Visualization of the algorithm

Complexity II

Pages have identical heights (after some page)

- Active list is bounded by a constant
- Thus
  - $O(n \times \langle\text{average length of active list}\rangle) = O(n)$

Pages have varying heights

- Active list can grow arbitrarily (i.e., $O(n)$)
- Thus we end up with
  - $O(n \times \langle\text{average length of active list}\rangle) = O(n^2)$

Spread and paragraph variations

- They add a factor of $O(1)$ to the length of the active list
- Thus the complexity doesn't change!

Visualization of the algorithm

Complexity II

Pages have identical heights (after some page)

- Active list is bounded by a constant
- Thus
  - $O(n \times \langle\text{average length of active list}\rangle) = O(n)$

Pages have varying heights

- Active list can grow arbitrarily (i.e., $O(n)$)
- Thus we end up with
  - $O(n \times \langle\text{average length of active list}\rangle) = O(n^2)$

Spread and paragraph variations

- They add a factor of $O(1)$ to the length of the active list
- Thus the complexity doesn't change!

Managing floats

John Tenniel, 1870

Managing floats

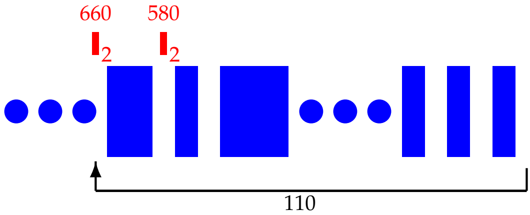
A visualization

We extend the algorithm at the point where we . . .

- . . . add active nodes for a new spread (here page 3)

Managing floats

A visualization

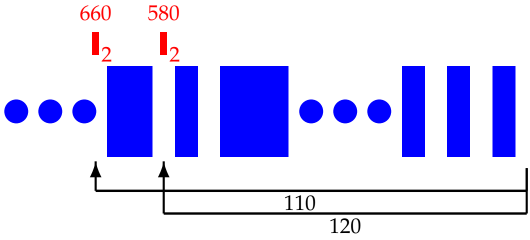


Step

- Trying to build page 3 (success) …

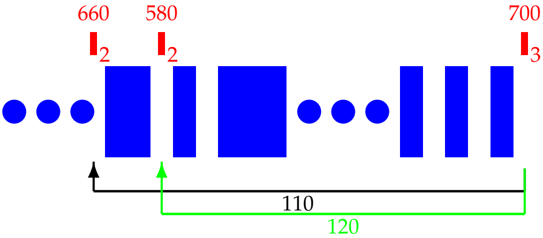
Managing floats

A visualization



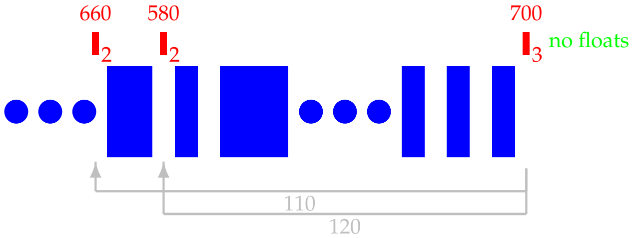
Step

- … an alternative (slightly higher costs) …



Step

- but overall better . . . so this ends the spread!

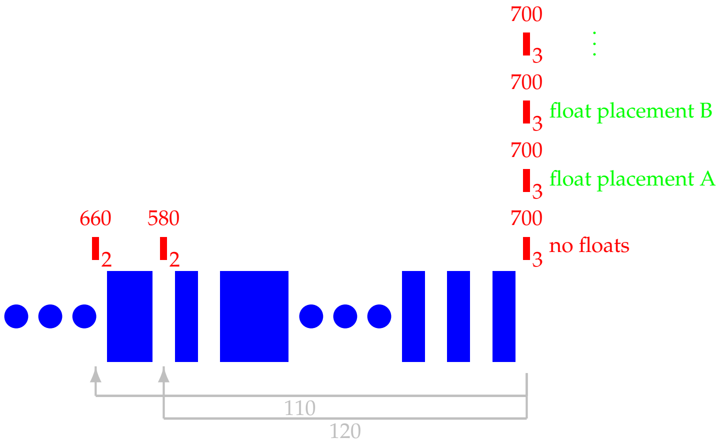


Now we prepare float placements for the next spread

- … this is for the case without floats

Managing floats

A visualization

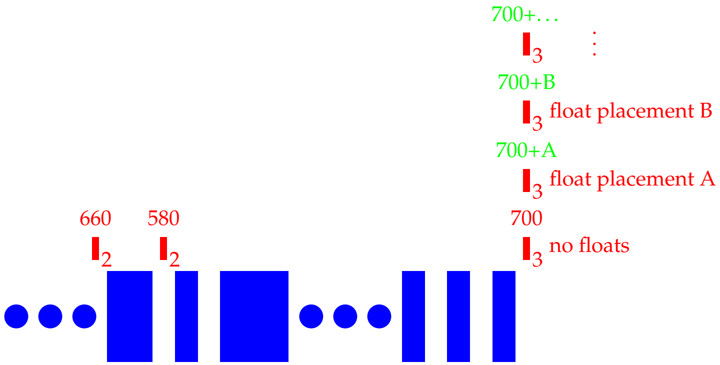


Now we prepare float placements for the next spread

- … and for each layout with floats add another node

Managing floats

A visualization

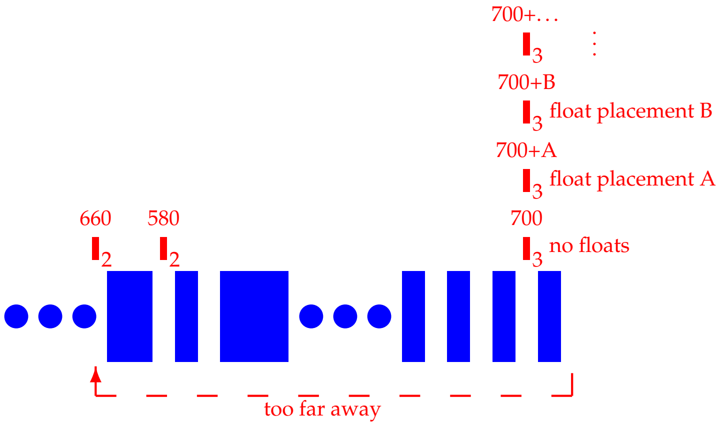


Now we prepare float placements for the next spread

- ... which may have extra costs associated ...

Managing floats

A visualization

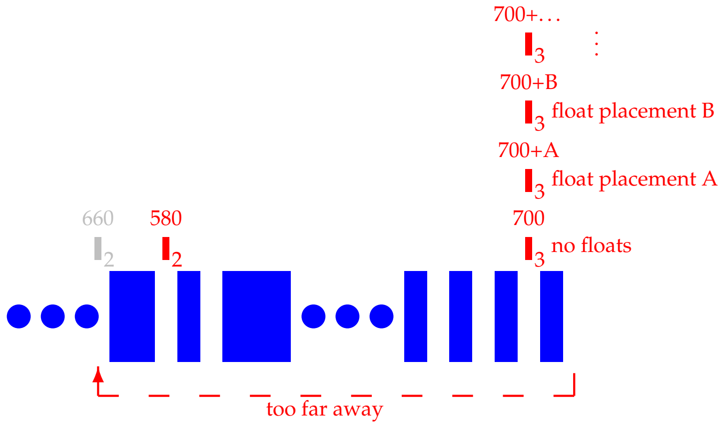


Step

- … then we continue looping …

Managing floats

A visualization

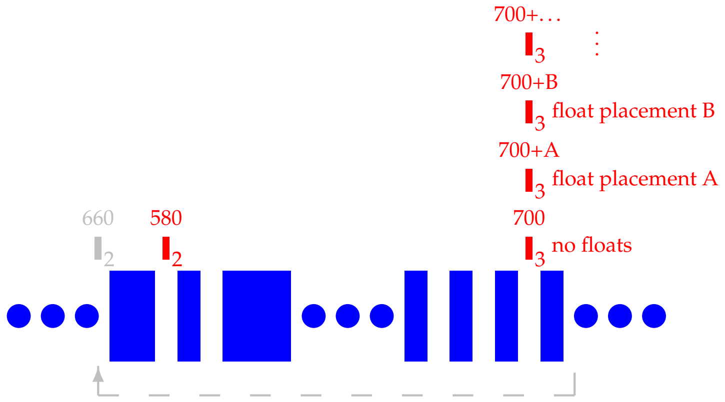


Step

- ... then we continue looping ...

Managing floats

A visualization



Step

- ... and so on and so forth ...

Executing the extended algorithm

Main points

When ending a spread

- Add new active nodes for each candidate placement
- Add "costs" in case the placement involves a preference rule that can be determined at this point (e.g., some float regions are preferred over other)

When seeing a call-out

- Check if any call-out/float relation is violated and if so deactivate the corresponding active node
- If a call-out/float preference rule is triggered we add the corresponding costs to the active node

When attempting to make a page (or column)

- Make a new active node only if we have seen all required call-outs (i.e., otherwise the attempt fails)

Executing the extended algorithm

Main points

When ending a spread

- Add new active nodes for each candidate placement
- Add "costs" in case the placement involves a preference rule that can be determined at this point (e.g., some float regions are preferred over other)

When seeing a call-out

- Check if any call-out/float relation is violated and if so deactivate the corresponding active node
- If a call-out/float preference rule is triggered we add the corresponding costs to the active node

When attempting to make a page (or column)

- Make a new active node only if we have seen all required call-outs (i.e., otherwise the attempt fails)

Executing the extended algorithm

Main points

When ending a spread

- Add new active nodes for each candidate placement
- Add “costs” in case the placement involves a preference rule that can be determined at this point (e.g., some float regions are preferred over other)

When seeing a call-out

- Check if any call-out/float relation is violated and if so deactivate the corresponding active node
- If a call-out/float preference rule is triggered we add the corresponding costs to the active node

When attempting to make a page (or column)

- Make a new active node only if we have seen all required call-outs (i.e., otherwise the attempt fails)

Executing the extended algorithm

Main points

When ending a spread

- Add new active nodes for each candidate placement
- Add "costs" in case the placement involves a preference rule that can be determined at this point (e.g., some float regions are preferred over other)

When seeing a call-out

- Check if any call-out/float relation is violated and if so deactivate the corresponding active node
- If a call-out/float preference rule is triggered we add the corresponding costs to the active node

When attempting to make a page (or column)

- Make a new active node only if we have seen all required call-outs (i.e., otherwise the attempt fails)

Executing the extended algorithm

Main points

When ending a spread

- Add new active nodes for each candidate placement
- Add “costs” in case the placement involves a preference rule that can be determined at this point (e.g., some float regions are preferred over other)

When seeing a call-out

- Check if any call-out/float relation is violated and if so deactivate the corresponding active node
- If a call-out/float preference rule is triggered we add the corresponding costs to the active node

When attempting to make a page (or column)

- Make a new active node only if we have seen all required call-outs (i.e., otherwise the attempt fails)

Executing the extended algorithm

Main points

When ending a spread

- Add new active nodes for each candidate placement
- Add "costs" in case the placement involves a preference rule that can be determined at this point (e.g., some float regions are preferred over other)

When seeing a call-out

- Check if any call-out/float relation is violated and if so deactivate the corresponding active node
- If a call-out/float preference rule is triggered we add the corresponding costs to the active node

When attempting to make a page (or column)

- Make a new active node only if we have seen all required call-outs (i.e., otherwise the attempt fails)

Precompute candidate float placements

(whenever a spread has ended)

Without any restricting rules (the bad case)

- We know which floats have already been placed
- There is only a limited amount of space available
- But beyond that: We know nothing!

Important facts resulting from the above

- Number of placements is $O(n^c)$ for some constant $c > 1$
  - c is roughly the average the number of floats that can be placed on a spread

Precompute candidate float placements

(whenever a spread has ended)

Without any restricting rules (the bad case)

- We know which floats have already been placed
- There is only a limited amount of space available
- But beyond that: We know nothing!

Important facts resulting from the above

- Number of placements is $O(n^c)$ for some constant $c > 1$
  - c is roughly the average the number of floats that can be placed on a spread

Precompute candidate float placements

(whenever a spread has ended)

Without any restricting rules (the bad case)

- We know which floats have already been placed
- There is only a limited amount of space available
- But beyond that: We know nothing!

Important facts resulting from the above

- Number of placements is $O(n^c)$ for some constant $c > 1$
  - c is roughly the average the number of floats that can be placed on a spread

Precompute candidate float placements

(whenever a spread has ended)

Without any restricting rules (the bad case)

- We know which floats have already been placed
- There is only a limited amount of space available
- But beyond that: We know nothing!

Important facts resulting from the above

- Number of placements is $O(n^c)$ for some constant $c > 1$
  - c is roughly the average the number of floats that can be placed on a spread

Precompute candidate float placements

(whenever a spread has ended)

Without any restricting rules (the bad case)

- We know which floats have already been placed
- There is only a limited amount of space available
- But beyond that: We know nothing!

Important facts resulting from the above

- Number of placements is $O(n^c)$ for some constant $c > 1$
  - c is roughly the average the number of floats that can be placed on a spread

Precompute candidate float placements

(whenever a spread has ended)

Without any restricting rules (the bad case)

- We know which floats have already been placed
- There is only a limited amount of space available
- But beyond that: We know nothing!

Important facts resulting from the above

- Number of placements is $O(n^c)$ for some constant $c > 1$
  - c is roughly the average the number of floats that can be placed on a spread

So this will get unmanageable fast!

Rule types

- Absolute rules: placement not allowed if violated
- Preference rules: placement is (un)favorable

Call-out / float constraints

- Floats are placed in order of their first/main call-out
  - Different streams are (usually) independent
- A float must appear after its call-out …
  - same or later column
  - strictly after
  - same page or spread or later
  - must be placed in their subsection
  - must be visible from the call-out

Float rules (structuring the approach)

Different types of rules

Rule types

- Absolute rules: placement not allowed if violated
- Preference rules: placement is (un)favorable

Call-out / float constraints

- Floats are placed in order of their first/main call-out
  - Different streams are (usually) independent
- A float must appear after its call-out …
  - same or later column (usual approach)
  - strictly after (fairly restrictive)
  - same page or spread or later (difficult with greedy algorithms; involves reformatting)
  - must be placed in their subsection (dangerous)
  - must be visible from the call-out (very dangerous)

Float rules (structuring the approach)

Different types of rules

Rule types

- Absolute rules: placement not allowed if violated
- Preference rules: placement is (un)favorable

Call-out / float constraints

- Floats are placed in order of their first/main call-out
  - Different streams are (usually) independent
- A float must appear after its call-out …
  - same or later column (usual approach)
  - strictly after (fairly restrictive)
  - same page or spread or later (difficult with greedy algorithms; involves reformatting)
  - must be placed in their subsection (dangerous)
  - must be visible from the call-out (very dangerous)

Float rules (structuring the approach)

Different types of rules

Rule types

- Absolute rules: placement not allowed if violated
- Preference rules: placement is (un)favorable

Call-out / float constraints

- Floats are placed in order of their first/main call-out
  - Different streams are (usually) independent
- A float must appear after its call-out ...
  - same or later column (usual approach)
  - strictly after (fairly restrictive)
  - same page or spread or later (difficult with greedy algorithms; involves reformatting)
  - must be placed in their subsection (dangerous)
  - must be visible from the call-out (very dangerous)

Float rules (structuring the approach)

Different types of rules

Rule types

- Absolute rules: placement not allowed if violated
- Preference rules: placement is (un)favorable

Call-out / float constraints

- Floats are placed in order of their first/main call-out
  - Different streams are (usually) independent
- A float must appear after its call-out ...
  - same or later column (usual approach)
  - strictly after (fairly restrictive)
  - same page or spread or later (difficult with greedy algorithms; involves reformatting)
  - must be placed in their subsection (dangerous)
  - must be visible from the call-out (very dangerous)

Float rules (structuring the approach)

Different types of rules

Rule types

- Absolute rules: placement not allowed if violated
- Preference rules: placement is (un)favorable

Call-out / float constraints

- Floats are placed in order of their first/main call-out
  - Different streams are (usually) independent
- A float must appear after its call-out …
  - same or later column (usual approach)
  - strictly after (fairly restrictive)
  - same page or spread or later (difficult with greedy algorithms; involves reformatting)
  - must be placed in their subsection (dangerous)
  - must be visible from the call-out (very dangerous)

Float rules (structuring the approach)

Different types of rules

Rule types

- Absolute rules: placement not allowed if violated
- Preference rules: placement is (un)favorable

Call-out / float constraints

- Floats are placed in order of their first/main call-out
  - Different streams are (usually) independent
- A float must appear after its call-out …
  - same or later column (usual approach)
  - strictly after (fairly restrictive)
  - same page or spread or later (difficult with greedy algorithms; involves reformatting)
  - must be placed in their subsection (dangerous)
  - must be visible from the call-out (very dangerous)

Float rules (structuring the approach)

Different types of rules

Rule types

- Absolute rules: placement not allowed if violated
- Preference rules: placement is (un)favorable

Call-out / float constraints

- Floats are placed in order of their first/main call-out
  - Different streams are (usually) independent
- A float must appear after its call-out …
  - same or later column (usual approach)
  - strictly after (fairly restrictive)
  - same page or spread or later (difficult with greedy algorithms; involves reformatting)
  - must be placed in their subsection (dangerous)
  - must be visible from the call-out (very dangerous)

Float rules (structuring the approach)

Different types of rules, continued

Column, page, spread related constraints

- Number of floats
  - Example: no more than x floats on top of column
  - Example: no more than y floats on spread
  - Example: more than one float per page is discouraged
- Area sizing
  - Example: minimum of x% of text required
  - Example: bottom area must be smaller than …
- Area relations
  - Example: only top or bottom area can be used
  - Example: Adjacent areas need visually compatible floats

Float rules (structuring the approach)

Different types of rules, continued

Column, page, spread related constraints

- Number of floats
  - Example: no more than x floats on top of column
  - Example: no more than y floats on spread
  - Example: more than one float per page is discouraged
- Area sizing
  - Example: minimum of x% of text required
  - Example: bottom area must be smaller than …
- Area relations
  - Example: only top or bottom area can be used
  - Example: Adjacent areas need visually compatible floats

Float rules (structuring the approach)

Different types of rules, continued

Column, page, spread related constraints

- Number of floats
  - Example: no more than x floats on top of column
  - Example: no more than y floats on spread
  - Example: more than one float per page is discouraged
- Area sizing
  - Example: minimum of x% of text required
  - Example: bottom area must be smaller than …
- Area relations
  - Example: only top or bottom area can be used
  - Example: Adjacent areas need visually compatible floats

Float rules (structuring the approach)

Different types of rules, continued

Column, page, spread related constraints

- Number of floats
  - Example: no more than x floats on top of column
  - Example: no more than y floats on spread
  - Example: more than one float per page is discouraged
- Area sizing
  - Example: minimum of x% of text required
  - Example: bottom area must be smaller than …
- Area relations
  - Example: only top or bottom area can be used
  - Example: Adjacent areas need visually compatible floats

Precompute candidate float placements
(whenever a spread has ended)

Floats are sequenced (the better case)

- We know which floats are next in sequence
- There is only a limited amount of space available
- We know if a call-out for a float can appear on the next spread

Important facts

- Number of placements is bounded by a constant
- Placements can be computed in linear time (and fast)

Precompute candidate float placements
(whenever a spread has ended)

Floats are sequenced (the better case)

- We know which floats are next in sequence
- There is only a limited amount of space available
- We know if a call-out for a float can appear on the next spread

Important facts

- Number of placements is bounded by a constant
- Placements can be computed in linear time (and fast)

Precompute candidate float placements

(whenever a spread has ended)

Floats are sequenced (the better case)

- We know which floats are next in sequence
- There is only a limited amount of space available
- We know if a call-out for a float can appear on the next spread

Important facts

- Number of placements is bounded by a constant
- Placements can be computed in linear time (and fast)

Precompute candidate float placements

(whenever a spread has ended)

Floats are sequenced (the better case)

- We know which floats are next in sequence
- There is only a limited amount of space available
- We know if a call-out for a float can appear on the next spread

Important facts

- Number of placements is bounded by a constant
- Placements can be computed in linear time (and fast)

Precompute candidate float placements

(whenever a spread has ended)

Floats are sequenced (the better case)

- We know which floats are next in sequence
- There is only a limited amount of space available
- We know if a call-out for a float can appear on the next spread

Important facts

- Number of placements is bounded by a constant
- Placements can be computed in linear time (and fast)

Precompute candidate float placements
(whenever a spread has ended)

Floats are sequenced (the better case)

- We know which floats are next in sequence
- There is only a limited amount of space available
- We know if a call-out for a float can appear on the next spread

Important facts

- Number of placements is bounded by a constant
- Placements can be computed in linear time (and fast)

Complexity with sequenced floats

Assumptions

- Floats are placed sequentially
- Different float streams are independent
- Relation between # of floats and the document length is linear

The length of the active list

- First float on the next spread may be any of the floats (i.e., the possibilities are equal to # of floats)
- Number of different candidate solutions with the first float fixed is bounded by a constant

The overall complexity is therefore

- $O(n^{\text{\# of float streams}+1})$ if the page height is fixed
- $O(n^{\text{\# of float streams}+2})$ otherwise

Complexity with sequenced floats

Assumptions

- Floats are placed sequentially
- Different float streams are independent
- Relation between # of floats and the document length is linear

The length of the active list

- First float on the next spread may be any of the floats (i.e., the possibilities are equal to # of floats)
- Number of different candidate solutions with the first float fixed is bounded by a constant

The overall complexity is therefore

- $O(n^{\text{\# of float streams}+1})$ if the page height is fixed
- $O(n^{\text{\# of float streams}+2})$ otherwise

Complexity with sequenced floats

Assumptions

- Floats are placed sequentially
- Different float streams are independent
- Relation between # of floats and the document length is linear

The length of the active list

- First float on the next spread may be any of the floats (i.e., the possibilities are equal to # of floats)
- Number of different candidate solutions with the first float fixed is bounded by a constant

The overall complexity is therefore

- $O(n^{\#\text{ of float streams}+1})$ if the page height is fixed
- $O(n^{\#\text{ of float streams}+2})$ otherwise

Complexity with sequenced floats

Assumptions

- Floats are placed sequentially
- Different float streams are independent
- Relation between # of floats and the document length is linear

The length of the active list

- First float on the next spread may be any of the floats (i.e., the possibilities are equal to # of floats)
- Number of different candidate solutions with the first float fixed is bounded by a constant

The overall complexity is therefore

- $O(n^{\text{\# of float streams}+1})$ if the page height is fixed
- $O(n^{\text{\# of float streams}+2})$ otherwise

Complexity with sequenced floats

Assumptions

- Floats are placed sequentially
- Different float streams are independent
- Relation between # of floats and the document length is linear

The length of the active list

- First float on the next spread may be any of the floats (i.e., the possibilities are equal to # of floats)
- Number of different candidate solutions with the first float fixed is bounded by a constant

The overall complexity is therefore

- $O(n^{\text{\# of float streams}+1})$ if the page height is fixed
- $O(n^{\text{\# of float streams}+2})$ otherwise

Complexity with sequenced floats

Assumptions

- Floats are placed sequentially
- Different float streams are independent
- Relation between # of floats and the document length is linear

The length of the active list

- First float on the next spread may be any of the floats (i.e., the possibilities are equal to # of floats)
- Number of different candidate solutions with the first float fixed is bounded by a constant

The overall complexity is therefore

- $O(n^{\#\text{ of float streams}+1})$ if the page height is fixed
- $O(n^{\#\text{ of float streams}+2})$ otherwise

The overall complexity — Anything non-linear is bad news

John Tenniel, 1870

Apply pruning

- Require that floats stay close to their call-out:
  - Candidate solutions that require too many page-turns are dropped
  - (unless floats cannot be placed earlier)

Is this adequate?

- Yes: Users expect to see a float close to its call-out
- Unnecessary page-turns reduce reading experience

The overall complexity — Anything non-linear is bad news

John Tenniel, 1870

Apply pruning

- Require that floats stay close to their call-out:
  - Candidate solutions that require too many page-turns are dropped
  - (unless floats cannot be placed earlier)

Is this adequate?

- Yes: Users expect to see a float close to its call-out
- Unnecessary page-turns reduce reading experience

The overall complexity — Anything non-linear is bad news

John Tenniel, 1870

Apply pruning

- Require that floats stay close to their call-out:
  - Candidate solutions that require too many page-turns are dropped
  - (unless floats cannot be placed earlier)

Is this adequate?

- Yes: Users expect to see a float close to its call-out
- Unnecessary page-turns reduce reading experience

The overall complexity — With pruning applied

Length of the active list

- First float on next spread must have its call-out close by
- Thus, the number of different possibilities for the first float is bounded by a constant
- Thus, the factor by which the active list can increase is bounded by a constant

The overall complexity drops back to

- $O(n)$ if the page height is fixed
- $O(n^2)$ otherwise

The overall complexity — With pruning applied

Length of the active list

- First float on next spread must have its call-out close by
- Thus, the number of different possibilities for the first float is bounded by a constant
- Thus, the factor by which the active list can increase is bounded by a constant

The overall complexity drops back to

- $O(n)$ if the page height is fixed
- $O(n^2)$ otherwise

The overall complexity — With pruning applied

Length of the active list

- First float on next spread must have its call-out close by
- Thus, the number of different possibilities for the first float is bounded by a constant
- Thus, the factor by which the active list can increase is bounded by a constant

The overall complexity drops back to

- $O(n)$ if the page height is fixed
- $O(n^2)$ otherwise

The overall complexity — With pruning applied

Length of the active list

- First float on next spread must have its call-out close by
- Thus, the number of different possibilities for the first float is bounded by a constant
- Thus, the factor by which the active list can increase is bounded by a constant

The overall complexity drops back to

- $O(n)$ if the page height is fixed
- $O(n^2)$ otherwise

The overall complexity — With pruning applied

Length of the active list

- First float on next spread must have its call-out close by
- Thus, the number of different possibilities for the first float is bounded by a constant
- Thus, the factor by which the active list can increase is bounded by a constant

The overall complexity drops back to

- $O(n)$ if the page height is fixed
- $O(n^2)$ otherwise

…which is where we want it to be

🙂

Time and space . . .

or what happens in real life

John Tenniel, 1870

“Well, in OUR country,” said Alice, still panting a little, “you’d generally get to somewhere else—if you ran very fast for a long time, as we’ve been doing.”

“A slow sort of country!” said the Queen. “Now, HERE, you see, it takes all the running YOU can do, to keep in the same place. If you want to get somewhere else, you must run at least twice as fast as that!”

The chosen challenge

John Tenniel, 1870

Through the Looking Glass

- A galley with 2516 breakpoints
- 47 pictures to place

The ask

- Layout with 46 lines per column
- No orphans and widows!
- Maximum of one figure per column
- Favor solutions with sections at column top

Trial results with standard LaTeX (i.e., greedy algorithm)

Always enforced / preferred

- No widows and orphans
- Minimum of 2 text lines after a heading
- Encourage headings at top of columns

Running time: less than 2 seconds

Results:

- 98 text columns
  - 55 good columns (badness < 4000)
  - 1 horrible column (badness 6559)
  - 34 infinitly bad columns
- 5 half-empty float columns
- Only 1 heading at top of column (out of 9)

Estimated time for fixing: $(35 + 5) \times 15\,\text{min} \approx 10\,\text{hours}$

Trial results with standard LaTeX (i.e., greedy algorithm)

Always enforced / preferred

- No widows and orphans
- Minimum of 2 text lines after a heading
- Encourage headings at top of columns

Running time: less than 2 seconds

Results:

- 98 text columns
  - 55 good columns (badness < 4000)
  - 1 horrible column (badness 6559)
  - 34 infinitly bad columns
- 5 half-empty float columns
- Only 1 heading at top of column (out of 9)

Estimated time for fixing: $(35+5) \times 15\,\text{min} \approx 10\,\text{hours}$

Trial results with standard LaTeX (i.e., greedy algorithm)

Always enforced / preferred

- No widows and orphans
- Minimum of 2 text lines after a heading
- Encourage headings at top of columns

Running time: less than 2 seconds

Results:

- 98 text columns
  - 55 good columns (badness < 4000)
  - 1 horrible column (badness 6559)
  - 34 infinitly bad columns
- 5 half-empty float columns
- Only 1 heading at top of column (out of 9)

Estimated time for fixing: $(35 + 5) \times 15\,\mathrm{min} \approx$ 10 hours

Trial results with standard LaTeX (i.e., greedy algorithm)

Always enforced / preferred

- No widows and orphans
- Minimum of 2 text lines after a heading
- Encourage headings at top of columns

Running time: less than 2 seconds

Results:

- 98 text columns
  - 55 good columns (badness < 4000)
  - 1 horrible column (badness 6559)
  - 34 infinitly bad columns
- 5 half-empty float columns
- Only 1 heading at top of column (out of 9)

Estimated time for fixing: $(35 + 5) \times 15\,\text{min} \approx$ 10 hours

Trial results with global optimization and no restrictions (other than sequencing)

Always enforced / preferred

- No widows and orphans
- Minimum of 2 text lines after a heading
- Encourage headings at top of columns

Algorithm

- Include both paragraph and spread variations

Trial results with global optimization and no restrictions
(other than sequencing)

Always enforced / preferred

- No widows and orphans
- Minimum of 2 text lines after a heading
- Encourage headings at top of columns

Algorithm

- Include both paragraph and spread variations

More than 10 hours processing time

Trial results with parameterized objective function

(Floats and base algorithm and pruning)

Always enforced / preferred

- No widows and orphans
- Minimum of 2 text lines after a heading
- Encourage headings at top of columns

Algorithm

- Just the base algorithm + floats
- Try pruning after x page turns to shorten time necessary

Trial results with parameterized objective function

(Floats and base algorithm and pruning)

Always enforced / preferred

- No widows and orphans
- Minimum of 2 text lines after a heading
- Encourage headings at top of columns

Algorithm

- Just the base algorithm + floats
- Try pruning after x page turns to shorten time necessary

Runs out of options to optimize near the beginning

Trial results with parameterized objective function

(Floats and variations and pruning)

Always enforced / preferred

- No widows and orphans
- Minimum of 2 text lines after a heading
- Encourage headings at top of columns

Restricting allowed page-turns

- Allow 0, 1, 2, 3, . . . turns (per float)
- Costs = expensive / moderate / cheap

Spread length variations

- Disallowed / expensive / moderate / cheap

Paragraph length variations (\looseness)

- Disallowed / allowed (costs based on paragraph quality)

Trial results with parameterized objective function

(Floats and variations and pruning)

Always enforced / preferred

- No widows and orphans
- Minimum of 2 text lines after a heading
- Encourage headings at top of columns

Restricting allowed page-turns

- Allow 0, 1, 2, 3, . . . turns (per float)
- Costs = expensive / moderate / cheap

Spread length variations

- Disallowed / expensive / moderate / cheap

Paragraph length variations (\looseness)

- Disallowed / allowed (costs based on paragraph quality)

Trial results with parameterized objective function

(Floats and variations and pruning)

Always enforced / preferred

- No widows and orphans
- Minimum of 2 text lines after a heading
- Encourage headings at top of columns

Restricting allowed page-turns

- Allow 0, 1, 2, 3, ... turns (per float)
- Costs = expensive / moderate / cheap

Spread length variations

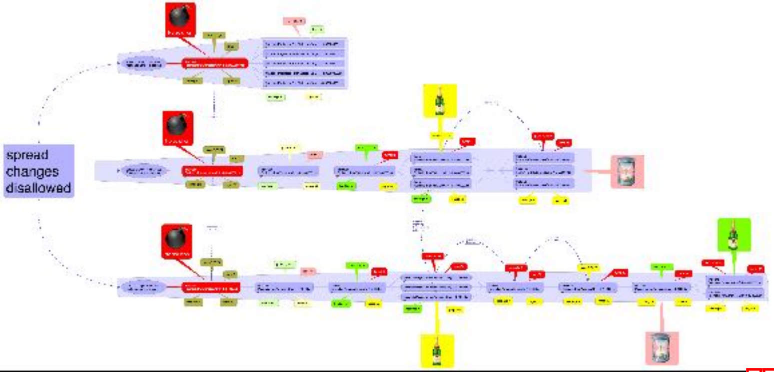
- Disallowed / expensive / moderate / cheap

Paragraph length variations (`\looseness`)

- Disallowed / allowed (costs based on paragraph quality)

Trial results with parameterized objective function
(Floats and paragraph variations)

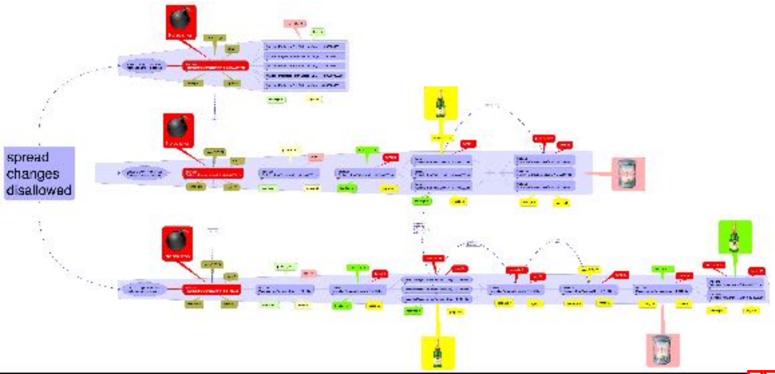
- Horizontally: apply pruning after 0, 1, 2, … page turns
- Vertically: page turn costs are expensive / moderate / cheap
- Champagne bottles indicate 8 or 9 sections placed on column top
- Hourglass means this (and later) trials need more than 5 min



Trial results with parameterized objective function

(Floats and paragraph variations)

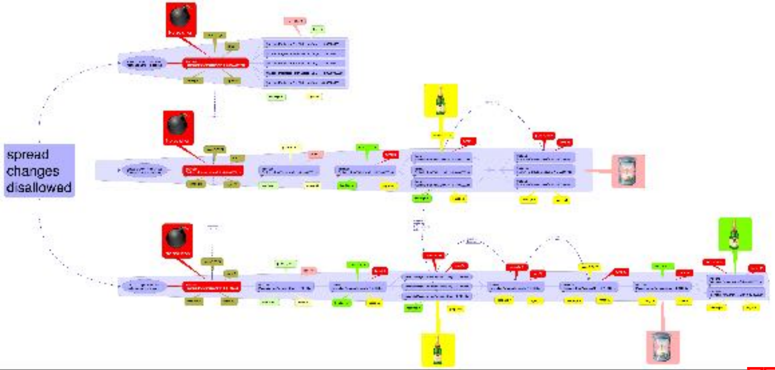
- Horizontally: apply pruning after 0, 1, 2, … page turns
- Vertically: page turn costs are expensive / moderate / cheap
- Champagne bottles indicate 8 or 9 sections placed on column top
- Hourglass means this (and later) trials need more than 5 min



Trial results with parameterized objective function

(Floats and paragraph variations)

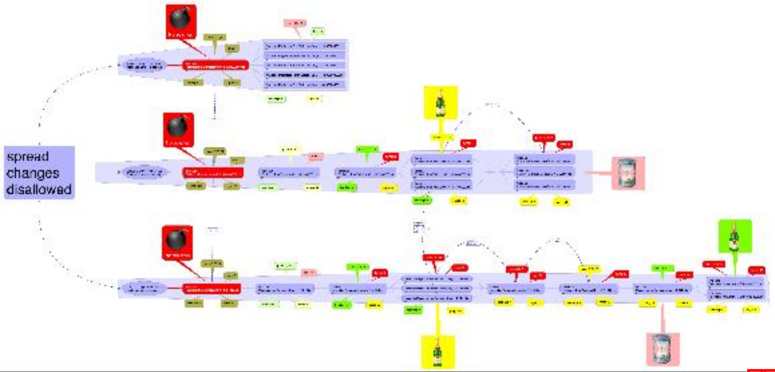
- Horizontally: apply pruning after 0, 1, 2, . . . page turns
- Vertically: page turn costs are expensive / moderate / cheap
- Champagne bottles indicate 8 or 9 sections placed on column top
- Hourglass means this (and later) trials need more than 5 min



Trial results with parameterized objective function

(Floats and paragraph variations)

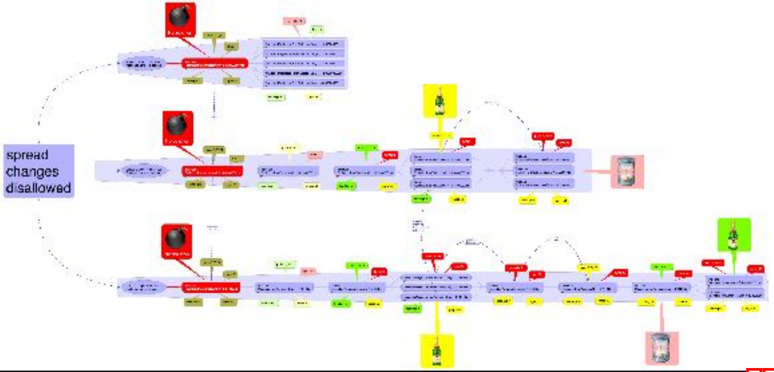
- Horizontally: apply pruning after 0, 1, 2, … page turns
- Vertically: page turn costs are expensive / moderate / cheap
- Champagne bottles indicate 8 or 9 sections placed on column top
- Hourglass means this (and later) trials need more than 5 min



Trial results with parameterized objective function

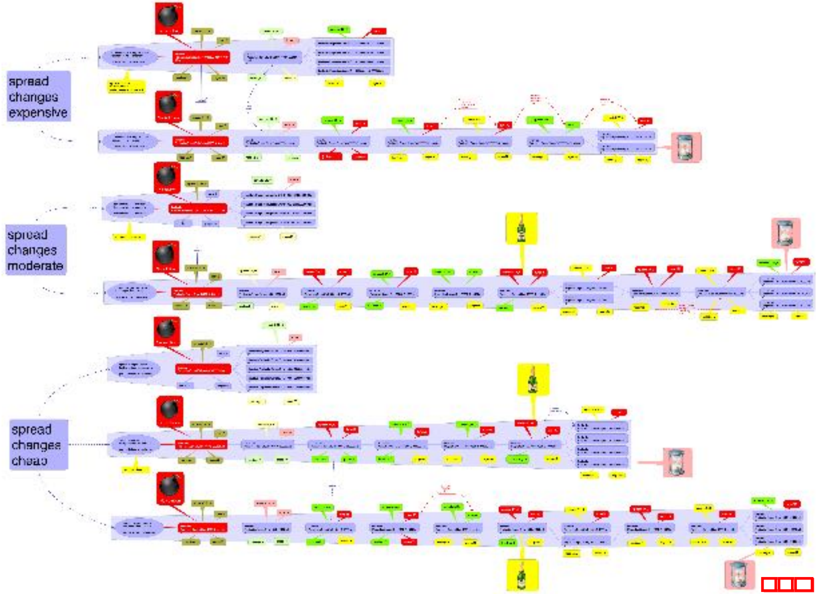
(Floats and paragraph variations)

- Horizontally: apply pruning after 0, 1, 2, ... page turns
- Vertically: page turn costs are expensive / moderate / cheap
- Champagne bottles indicate 8 or 9 sections placed on column top
- Hourglass means this (and later) trials need more than 5 min



Trial results with parameterized objective function

(Floats and spread variations)



Trial results with parameterized objective function

(Floats and all variations)

Summary of results
(when using floats)

Pagination with greedy algorithm still fails for nearly all cases (i.e., floats do not make things better)

Idea: use dynamic programming approach for pagination with extra flexibility and add floats

Doable … in that case complexity is

- $O(n^{\text{\# of float streams}+1})$ if the page height is fixed
- $O(n^{\text{\# of float streams}+2})$ otherwise

But …

- quadratic or cubic growth (or worse) is still too slow
- who wants to wait 10+ hours each run?

Apply pruning … to cut down the search space

- this is reasonable as it fits with user expectations
- and produces results in acceptable time!

Summary of results

(when using floats)

Pagination with greedy algorithm still fails for nearly all cases (i.e., floats do not make things better)

Idea: use dynamic programming approach for pagination with extra flexibility and add floats

Doable … in that case complexity is

- $O(n^{\text{\# of float streams}+1})$ if the page height is fixed
- $O(n^{\text{\# of float streams}+2})$ otherwise

But …

- quadratic or cubic growth (or worse) is still too slow
- who wants to wait 10+ hours each run?

Apply pruning … to cut down the search space

- this is reasonable as it fits with user expectations
- and produces results in acceptable time!

Summary of results

(when using floats)

Pagination with greedy algorithm still fails for nearly all cases (i.e., floats do not make things better)

Idea: use dynamic programming approach for pagination with extra flexibility and add floats

Doable …in that case complexity is

- $O(n^{\#\text{ of float streams}+1})$ if the page height is fixed
- $O(n^{\#\text{ of float streams}+2})$ otherwise

But …

- quadratic or cubic growth (or worse) is still too slow
- who wants to wait 10+ hours each run?

Apply pruning … to cut down the search space

- this is reasonable as it fits with user expectations
- and produces results in acceptable time!

Summary of results

(when using floats)

Pagination with greedy algorithm still fails for nearly all cases (i.e., floats do not make things better)

Idea: use dynamic programming approach for pagination with extra flexibility and add floats

Doable ... in that case complexity is

- $O(n^{\#\text{ of float streams}+1})$ if the page height is fixed
- $O(n^{\#\text{ of float streams}+2})$ otherwise

But ...

- quadratic or cubic growth (or worse) is still too slow
- who wants to wait 10+ hours each run?

Apply pruning ... to cut down the search space

- this is reasonable as it fits with user expectations
- and produces results in acceptable time!

Summary of results

(when using floats)

Pagination with greedy algorithm still fails for nearly all cases (i.e., floats do not make things better)

Idea: use dynamic programming approach for pagination with extra flexibility and add floats

Doable … in that case complexity is

- $O(n^{\text{\# of float streams}+1})$ if the page height is fixed
- $O(n^{\text{\# of float streams}+2})$ otherwise

But …

- quadratic or cubic growth (or worse) is still too slow
- who wants to wait 10+ hours each run?

Apply pruning … to cut down the search space

- this is reasonable as it fits with user expectations
- and produces results in acceptable time!

Summary of results

(when using floats)

Pagination with greedy algorithm still fails for nearly all cases (i.e., floats do not make things better)

Idea: use dynamic programming approach for pagination with extra flexibility and add floats

Doable … in that case complexity is

- $O(n^{\text{\# of float streams}+1})$ if the page height is fixed
- $O(n^{\text{\# of float streams}+2})$ otherwise

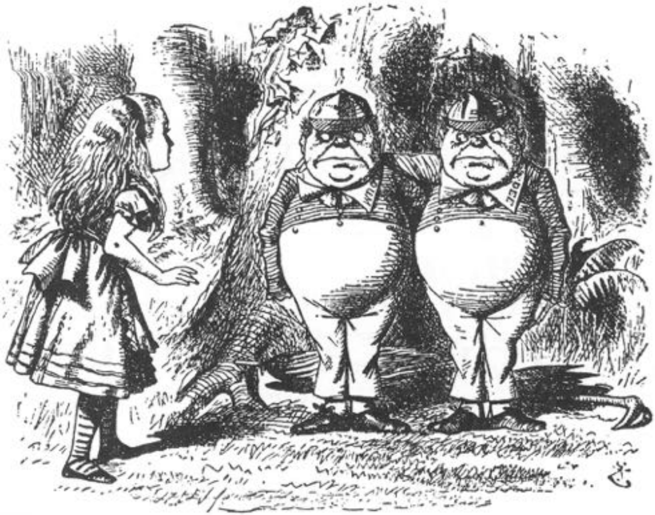
But …

- quadratic or cubic growth (or worse) is still too slow
- who wants to wait 10+ hours each run?

Apply pruning … to cut down the search space

- this is reasonable as it fits with user expectations
- and produces results in acceptable time!

Comparisons

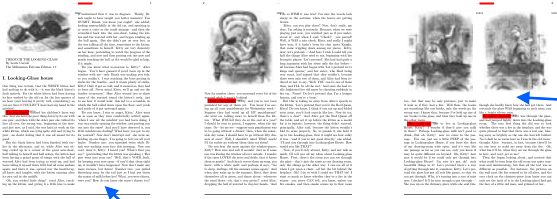


John Tenniel, 1870

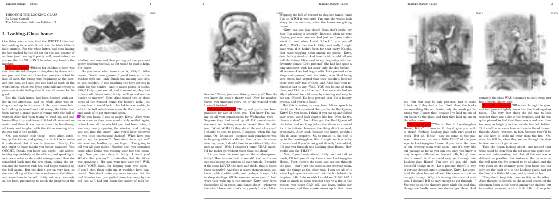
Comparison – spreads 1 + 2

greedy viz. optimal, 0 turns

greedy

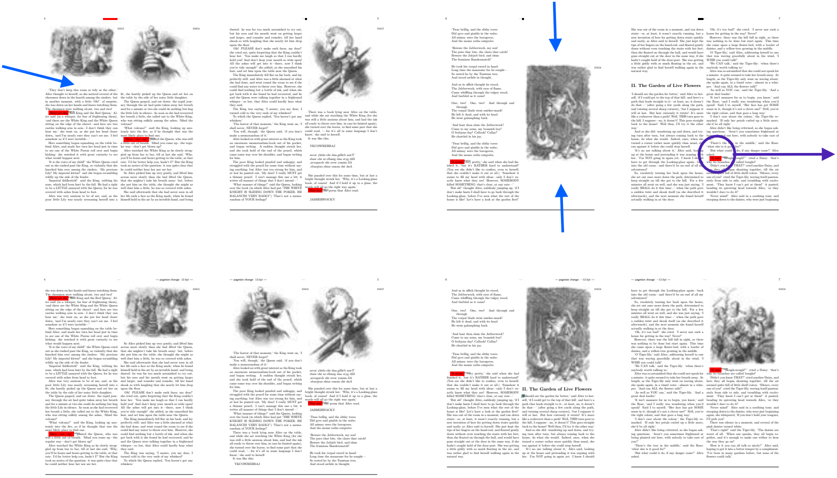


optimal



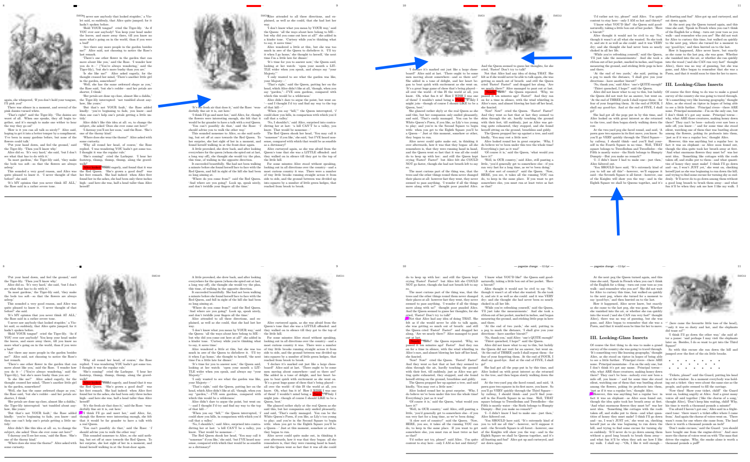
Comparison – spreads 3 + 4

greedy viz. optimal, 0 turns



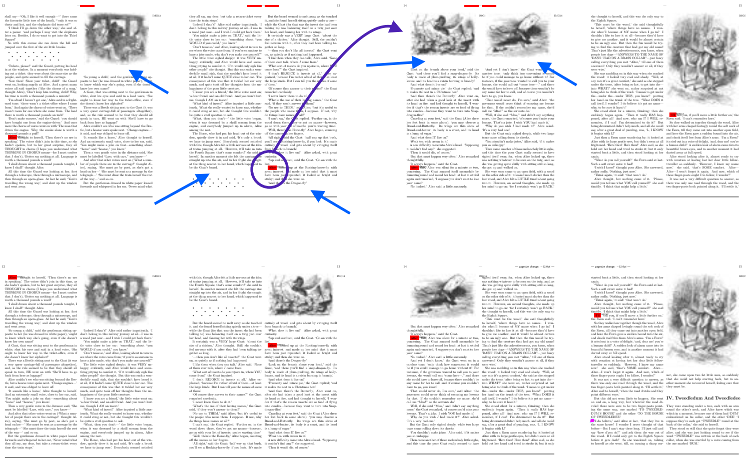
Comparison – spreads 5 + 6

greedy viz. optimal, 0 turns



Comparison – spreads 7 + 8

greedy viz. optimal, 0 turns

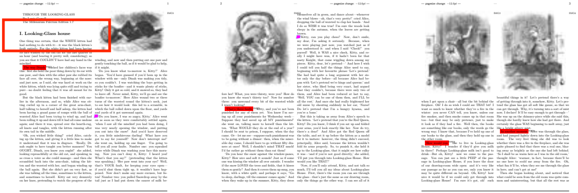


Comparison – spreads 1 + 2

optimal, 0 turns viz. optimal, 2 turns

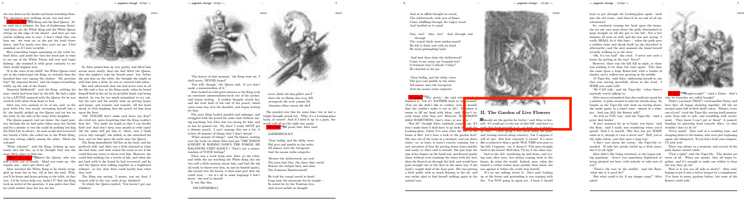
0 turns

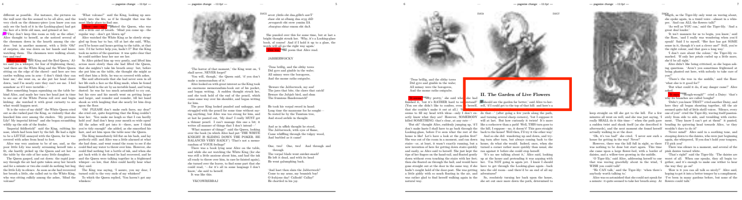
2 turns



Comparison – spreads 3 + 4

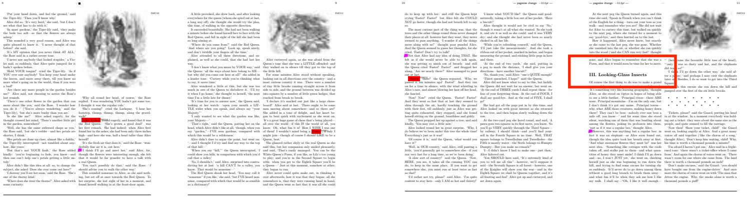
optimal, 0 turns viz. optimal, 2 turns

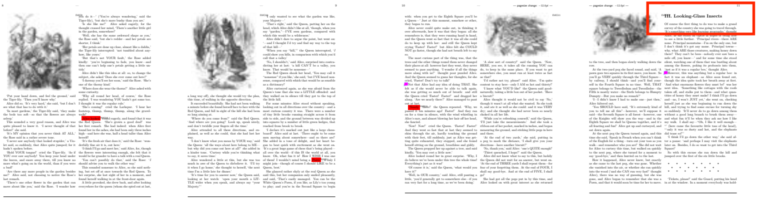




Comparison – spreads 5 + 6

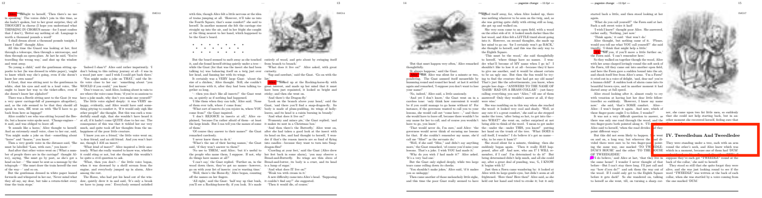
optimal, 0 turns viz. optimal, 2 turns





Comparison – spreads 7 + 8

optimal, 0 turns viz. optimal, 2 turns



Mischief managed!

Hope I was able to reveal something new for you.
Thank you!

John Tenniel, 1870