Through the Looking Glass
… and what Alice found there

TUG Conference 2017, Bachotek

Frank Mittelbach
frank.mittelbach@latex-project.org

April 29, 2017

Contents

1 From the looking glass story

Time and space

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!"

Time and space

Placing figures on pages (general formula)

$$\binom{\text{pages} + \text{figures} - 1}{\text{figures}} = ?$$

Placing figures on pages (one per page maximum)

$$\binom{\text{pages}}{\text{figures}} = ?$$

Examples (assuming 1 second per quality assessment)

- 16 pages, 9 figures $\rightarrow 11440$ trials $\rightarrow 3.1$ hours
- 90 pages, 28 figures $\rightarrow 1.548 \times 10^{23}$ trials $\rightarrow 4.91 \times 10^{15}$ years

---

So what now?

Can it be helped a little?

Takayama 2009

2 The dynamic programming methodology

Defining the problem

Input model

- A sequence of text blocks $T = \{t_1, t_2, \dots, t_n\}$
- A sequence of (figure) floats $F = \{f_1, f_2, \dots, f_\ell\}$
- (possibly some more float sequences — ignored for now)

Layout model

- A sequence of spreads $\mathsf{S}_1, \mathsf{S}_2, \dots, \mathsf{S}_k$
  - with columns/pages (sizes may differ)
  - with areas for floats
  - constraints for the filling process
  - some further auxiliary information

Defining the problem (continued)

Paginations

- A mapping $p : T \cup F \to \{1, 2, \dots, k\}$ such that

$$p(t_i) \leq p(t_j) \qquad \text{for } 1 \leq i < j \leq n$$
$$p(f_i) \leq p(f_j) \qquad \text{for } 1 \leq i < j \leq \ell$$

- $\mathcal{P}$ is the set of all possible paginations of $T \cup F$

Objective function (cost function)

- A function $Q : \mathcal{P} \to \Re$

Optimization task

- We seek: $p_0 \in \mathcal{P}$ such that $Q(p_0) \leq Q(p)$ for all $p \in \mathcal{P}$

What can we do?
(Getting requirements for Q)

Example 1: Make a gut decision

- I.e., look at each pagination (for a second) and decide
- Clearly not workable:
  - Already for "Through the Looking Glass" that takes longer than the current age of the universe

Example 2: Base decision on call-out/float distance

- I.e., how many pages do I need to turn to reach a float
  - Linear formula: solvable using dynamic programming
  - Quadratic formula: NP-complete as shown by Plass

Example 3: Recto/verso criteria

- E.g., penalize if call-out and float are on the same type of page
  - Again NP-complete as shown by Plass

The dynamic programming methodology
When possible?

Problem consists of overlapping subproblems

- Clearly, that's the case (with sensible subproblems)
- We denote with $\mathcal{P}_{(\mathsf{S}_i,\ldots,\mathsf{S}_j)}{}^{a,b}_{c,d}$ to mean
  - all paginations of text blocks $t_a, \ldots, t_b$ and figures $f_c, \ldots, f_d$ onto spreads $\mathsf{S}_i, \ldots, \mathsf{S}_j$
- Examples:

$$\begin{aligned}
\mathcal{P}_{(\mathsf{S}_1,\mathsf{S}_2)}{}^{1,80}_{1,2} \times \mathcal{P}_{(\mathsf{S}_3,\mathsf{S}_4)}{}^{81,150}_{3,4} &\subset \mathcal{P}_{(\mathsf{S}_1,\ldots,\mathsf{S}_4)}{}^{1,150}_{1,4} \\
\mathcal{P}_{(\mathsf{S}_1,\ldots,\mathsf{S}_3)}{}^{1,110}_{1,2} \times \mathcal{P}_{(\mathsf{S}_4)}{}^{111,150}_{3,4} &\subset \mathcal{P}_{(\mathsf{S}_1,\ldots,\mathsf{S}_4)}{}^{1,150}_{1,4} \\
\mathcal{P}_{(\mathsf{S}_1,\mathsf{S}_2)}{}^{1,80}_{1,2} \times \mathcal{P}_{(\mathsf{S}_3)}{}^{81,110}_{\emptyset,\emptyset} &\subset \mathcal{P}_{(\mathsf{S}_1,\ldots,\mathsf{S}_3)}{}^{1,110}_{1,2}
\end{aligned}$$

Problem exhibits optimal substructure (optimality principle)

- The tricky bit
- A problem exhibits optimal substructure if
  - the optimal solution to the problem incorporates only optimal solutions to its subproblems;
  - the subproblems can be solved independently.
- Now what does this mean?

The dynamic programming methodology
Optimality principle

What does it mean?

- Assume we search for p_0 with $Q(p_0)$ minimal and

$$p_0 \in \mathcal{P}_{(\mathsf{S}_1,\ldots,\mathsf{S}_4)}{}^{1,150}_{1,4}$$

- Assume further that we find

$$p_0 \in \mathcal{P}_{(\mathsf{S}_1)}{}^{1,35}_{1,1} \times \mathcal{P}_{(\mathsf{S}_2)}{}^{36,80}_{2,2} \times \mathcal{P}_{(\mathsf{S}_3)}{}^{81,110}_{\emptyset,\emptyset} \times \mathcal{P}_{(\mathsf{S}_4)}{}^{111,150}_{3,4}$$

. . . then the optimality principle means that

- p_0 (suitably restricted) is also an optimal solution for

$$\mathcal{P}_{(\mathsf{S}_1)}{}^{1,35}_{1,1} \qquad \mathcal{P}_{(\mathsf{S}_1,\mathsf{S}_2)}{}^{1,80}_{1,2} \qquad \mathcal{P}_{(\mathsf{S}_1,\ldots,\mathsf{S}_3)}{}^{1,110}_{1,2}$$

- and many others, e.g., $\mathcal{P}_{(\mathsf{S}_2,\ldots,\mathsf{S}_4)}{}^{36,150}_{2,4}$ etc.

The dynamic programming methodology
Applying it

If dynamic programming is applicable we can

- solve each subproblem only once
- and remember the result
- construct the optimal solution of a bigger subproblem by extending and combining smaller subproblems

Example:

- Find the best way to put $t_1, \ldots, t_b$ and $f_1, \ldots, f_d$ onto spreads $\mathsf{S}_1, \ldots, \mathsf{S}_i$:

$$\mathcal{P}_{(\mathsf{S}_1,\ldots,\mathsf{S}_i)}{}^{1,b}_{1,d}$$

$$\begin{aligned}
\mathcal{P}_{(\mathsf{S}_1,\ldots,\mathsf{S}_i)}{}^{1,b}_{1,d} &\supset \mathcal{P}_{(\mathsf{S}_1,\ldots,\mathsf{S}_{i-1})}{}^{1,a}_{1,c} \times \mathcal{P}_{(\mathsf{S}_i)}{}^{a+1,b}_{c+1,d} \\
\mathcal{P}_{(\mathsf{S}_1,\ldots,\mathsf{S}_i)}{}^{1,b}_{1,d} &\supset \mathcal{P}_{(\mathsf{S}_1,\ldots,\mathsf{S}_{i-1})}{}^{1,a'}_{1,c'} \times \mathcal{P}_{(\mathsf{S}_i)}{}^{a'+1,b}_{c'+1,d} \\
&\vdots
\end{aligned}$$

The dynamic programming methodology
Applying it

Example continued:

- In other words, we have

$$\mathcal{P}_{(\mathsf{S}_1,\ldots,\mathsf{S}_i)}{}^{1,b}_{1,d} = \bigcup_{\substack{1 \le a \le b \\ 1 \le c \le d}} \mathcal{P}_{(\mathsf{S}_1,\ldots,\mathsf{S}_{i-1})}{}^{1,a}_{1,c} \times \mathcal{P}_{(\mathsf{S}_i)}{}^{a+1,b}_{c+1,d}$$

- So if we know the best way for each

$$\mathcal{P}_{(\mathsf{S}_1,\ldots,\mathsf{S}_{i-1})}{}^{1,a}_{1,c}$$

  then all we need to do is to calculate all the

$$\mathcal{P}_{(\mathsf{S}_i)}{}^{a+1,b}_{c+1,d}$$

  and apply Q to determine the best solution.

The dynamic programming methodology

Why does it sometimes fail?

Example continued:

- Suppose we have a pagination $p = p' \times p''$ with

$$p \in \mathcal{P}_{(\mathsf{S}_1,\ldots,\mathsf{S}_i)}{}^{1,b}_{1,d}$$
$$\text{and} \qquad p' \in \mathcal{P}_{(\mathsf{S}_1,\ldots,\mathsf{S}_{i-1})}{}^{1,a}_{1,c} \qquad p'' \in \mathcal{P}_{(\mathsf{S}_i)}{}^{a+1,b}_{c+1,d}$$

- Then we need to be able to calculate $Q(p)$ from $Q(p')$ and $Q(p'')$
- For example: $Q(p) = Q(p') + Q(p'') + \tilde{Q}(\mathcal{P}_{(\mathsf{S}_1,\ldots,\mathsf{S}_{i-1})}{}^{1,a}_{1,c})$

But for the NP-complete cases this is

- not possible as the "quality" depends on where the call-out is (within p') in relation to the float (in p'')
- not depending on a fixed value based on $\mathcal{P}_{(\mathsf{S}_1,\ldots,\mathsf{S}_{i-1})}{}^{1,a}_{1,c}$

3 The algorithms

The basic algorithm

(no floats)

Start up

- Let $A = \{a_0, a_1, a_2, \ldots\}$ be elements from the text stream that have been identified as places where we can end a spread (plus info how we got there)
- Initially this contains just a_0 (start of document)

Main loop through all elements $t^* \in T$

- Check if we can successfully build a spread from one or more $a \in A$ to the current t^*
- For each new spread that ends, check which path gives the best result (according to Q) and add that one as a new element to A
  - (here we need the optimality principle)
- Whenever some $a_i \in A$ is too far away from t^* (overfull spread) remove it from A

The basic algorithm continued

(no floats)

Finishing off

- Eventually, we will reach the end of the document …
- … then work from the best solution backwards through all the elements we passed through
- That defines our optimal solution

Complexity

- The outer loop has n elements
- The inner loop is the size of A which is
  - bounded by a constant if all spreads have the same structure $\rightarrow O(c \cdot n) = O(n)$
  - otherwise it can be at most n $\rightarrow O(n^2)$

The extended algorithm

(with floats)

When starting up

- Compile info about each call-out

When t^* is identified as a new endpoint for a spread

- Prepare a list of all possible float placements for the next spread (conservative)
- Add a new $a \in A$ for each of them

When finishing off

- We need to deal with the case of unplaced floats
  - We can, for example, add them on further spreads (with some extra costs)
  - or drop them as "non-solutions"

The extended algorithm continued

(with floats)

Complexity

- The outer loop has n elements
- The inner loop is the size of A:
  - The number of elements ending in a different t^* is either
    - $O(n)$ for fixed spread structure
    - or $O(n^2)$ otherwise
  - For each new t^* we compile the set of all potentially possible float placements for the next spread
    - This number is bounded by a constant (available space!)
    - Any of the available floats might be the first
- Thus
  - If the spread all have the same structure $\rightarrow O(n \cdot \ell)$
  - otherwise $\rightarrow O(n^2 \cdot \ell)$
- Floats add a complexity factor in the size of their stream!

Float rules (structuring the approach)

Different types of rules

Rule types

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

Call-out / float relations

- 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

Rules for placement

- There cannot be more than x floats on a single page
- The top area of a column may receive a maximum of y floats, the bottom area of z floats
- If more than $x\%$ of the space on a column is occupied by floats then no normal text will appear in that column
- Every column must contain a minimum of $x\%$ of text
- All the floats are stacked vertically vertically at the top of a page; alternative: they can appear at the top or bottom (but not in both places)
- Floats can be horizontally placed if they are visually compatible (e.g., have identical heights); might also be requested for floats placed in adjacent columns

Float rules (structuring the approach)

Different types of rules, continued

Rules for the inner structure of a float

- Position of caption/legend based on float size
- Position of caption/legend based on placement
- Float size alterations (cropping of graphics, etc.)

Pruning (dropping supposedly bad solutions)

- Too many unplaced floats and x previous columns have no floats allocated
  - But documents may have many call-outs close by (danger to drop too much) But only if the floats could have placed there (difficult to check)
- Distance between call-out and float too large
  - Described this way creates dependencies between subproblems, thus violate the optimality principle (difficult to implement correctly)
- Other ideas . . .
  - Topic for further research!

Applying float rules ...

Evaluate when deciding next float placement

- Pruning:
  - drop as soon as possible
- Absolute rules (for a spread):
  - drop if violated
- Preference rules (for a spread):
  - add cost charge

Evaluate when adding a call-out to a trial placement

- Call-out constraint rules (absolute):
  - remove $a \in A$ if violated
- Call-out constraint rules (preference):
  - add cost charge to $a \in A$

4 Aesthetics only

Designs without call-out constraints

(A bit of a horror scenario)

John Tenniel, 1870

Designs without call-out constraints

(A bit of a horror scenario)

What does this mean?

- No rules that favor a certain region (such as low distance from the call-out)
- The objective function only implements local aesthetics
- Thus the placement of floats mainly affects the quality through a better or worse fit of the text blocks

Consequences

- Dynamic programming would still work, as we can interpret this as the case in which
  - all call-outs are at the beginning of the document
  - the objective function adds a zero cost for the distance from the call-out
- But that means that pruning not really possible (what would be the criteria?)

Designs without call-out constraints

Managing the complexity

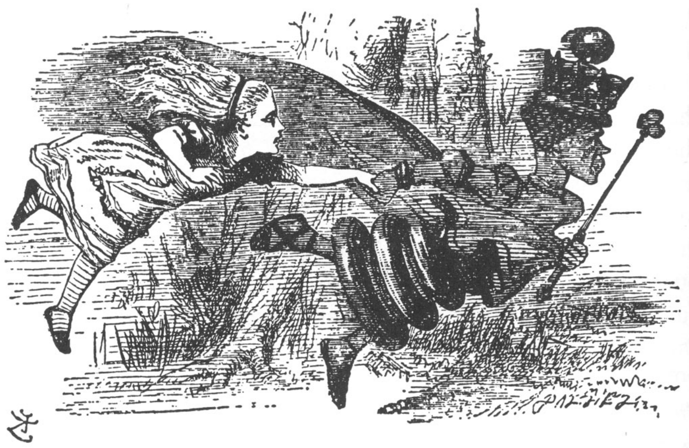
Just do it externally

- Advantage: fast
- Disadvantage: no interaction with formatting the text

Guiding the placement

- Advantage: interaction with text placement (while still fairly fast)
- Disadvantage: difficult to control
- More research necessary!

Mischief managed!



Hope I was able to reveal something new for you.
Thanks all around!

John Tenniel, 1870