June 1981

Report. No. STAN-CS-81-870

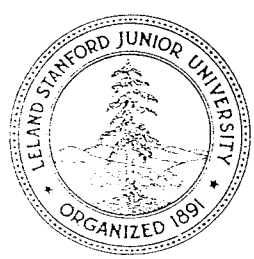
Optimal Pagination Techniques for Automatic Typesetting Systems

by

Michael F. Plass

Department of Computer Science

Stanford University
Stanford, CA 94305



Optimal Pagination Techniques for Automatic Typesetting Systems

Michael F. Plass

Computer Science Department
Stanford University
Stanford, California 94305

This report reproduces a dissertation submitted to the Department of Computer Science and the Committee on Graduate Studies of Stanford University in partial fulfillment of the requirements for the degree of Doctor of Philosophy. It is also available as Xerox technical report ISL-81-1.

Key words and phrases: Composition, Dynamic programming, Layout, NP-Completeness, Pagination, Shortest paths, Typesetting.

This research and printing was supported in part by National Science Foundation grants MCS-77-23738 and IST-7921977, and by Xerox Corporation.

Optimal Pagination Techniques for Automatic Typesetting Systems

A DISSERTATION
SUBMITTED TO THE
DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Michael Frederick Plass
June 1981

OPTIMAL PAGINATION TECHNIQUES
FOR AUTOMATIC TYPESETTING SYSTEMS

Michael Frederick Plass, PhD
Stanford University, 1981

Abstract: This thesis considers how to use a computer to break a document into pages suitable for printing. Although this problem is easy to solve when the document consists of just text, it becomes complicated when footnotes, displays, and figures are introduced. These elements add some freedom of choice in the way breaks are chosen, since the white space around the displays and the exact placement of the figures can be decided by the pagination algorithm. Out of the many possible ways to paginate such a document, the pagination algorithm should pick the one that is in some sense optimal. The approach taken here is to define a badness function that depends on the way the document is broken up, and then to design an algorithm to find a way to minimize the value of this function.

The document is modelled by two lists, the text list and the figure list. Each item in the text list is either a 'box', corresponding to something that will print such as a line of text, a 'glue' item, corresponding to the white space between the lines, a 'penalty' item, corresponding to a legal place to break the list, or a 'citation', marking a reference to one of the figures. The items in the figure list indicate the size of each figure, and by how much each figure is allowed to stretch or shrink. This model is based on the one used in the TEX typesetting system.

The optimizing pagination algorithm uses dynamic programming to find, for each i, j, and k, the best way to put the first i lines of text and the first j figures onto the first k pages; to make the program run in a reasonable amount of time, this calculation includes only those subproblems that are feasible, i.e., likely to lead to a solution with a small badness value.

The badness function must be chosen carefully in order to get a problem that can be solved by these techniques. For certain simple badness functions, the pagination problem is NP-complete; two such functions are described in the thesis.

Acknowledgements

THE GREATEST DEBT of gratitude that I owe concerning this thesis is to my advisor, Donald Knuth. In addition to conscientiously reading the draft and making hundreds of suggestions for its improvement, he made it all possible in the first place by making typesetting into a valid topic for research. And without TEX, the typesetting of the thesis in its present form would not have been possible. For this, and for his unerring advice while I worked under him, I give him my wholehearted thanks.

My thanks, also, to the other members of my reading committee, Luis Trabb Pardo, Andrew C. Yao, and Leo Guibas, who also made thoughtful comments on the contents of the thesis.

My wife, Susan, I thank for her continuing encouragement and for finding the quotation for the last chapter.

And I would also like to thank my parents for always encouraging my academic achievements, while never putting pressure on me to succeed.

Contents

List of Illustrations

Chapter 1

Introduction

There is nothing on earth more exquisite than a bonny book, with well-placed columns of rich black writing in beautiful borders, and illuminated pictures cunningly inset. But nowadays, instead of looking at books, people read them. — G. B. Shaw

When computers were first applied to typesetting two decades ago, they were programmed to handle the routine decisions, leaving the exceptional cases to be handled by a human. This is still true today. However, the art of developing these systems has progressed, and things that used to be exceptional are now considered to be routine. As an example, early systems could break paragraphs into lines, but they needed help from a human to hyphenate words when required. Later systems could hyphenate, but frequently failed to find a way to hyphenate a word or hyphenated erroneously, thus requiring intervention. Today's better systems need only occasional help for hyphenation. Pagination, the process of breaking a document into pages suitable for printing, is analogous to line breaking in many ways; however, it can be more complicated. For simple text, as in a novel, for example, pagination is trivial. However, as the text becomes more complex with the introduction of footnotes, displayed formulas, tables, diagrams, and illustrations, pagination becomes much more difficult, to the extent that every present-day typesetting system needs occasional help to perform this task satisfactorily. The thrust of the present work is to investigate ways of making pagination more routine, so that the computer can almost always do a satisfactory job, leaving comparatively few exceptional cases to be cleaned up by some overworked human.

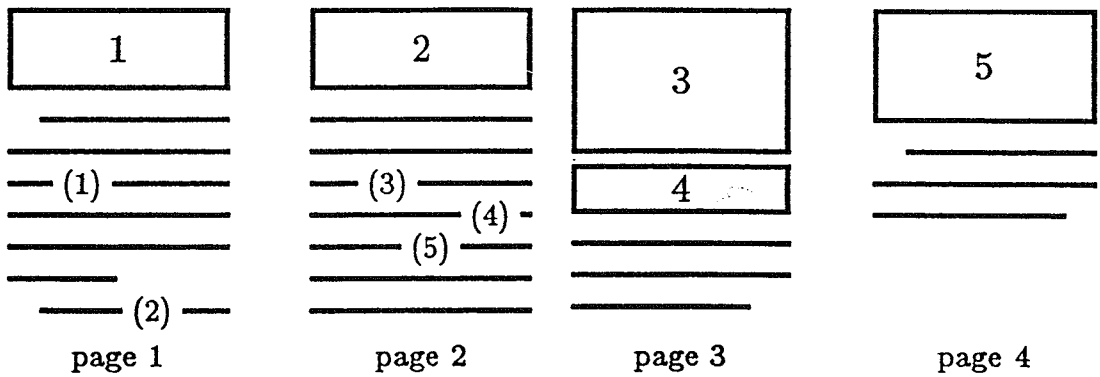


Figure 1. One way to paginate a little book.

As an example of the kinds of complexities that can arise in pagination, consider figure 1. The horizontal lines in this figure represent lines of text, and the boxes with numbers inside represent illustrations. The numbers in parentheses indicate where in the text the illustrations are cited. This must be some sort of children's book, because there is only room for ten lines of text on each page, and there are many illustrations for a book of this size. So imagine a second-grader sitting down to read this booklet. On the first page, she reads the part where the book talks about the first picture, and there it is, right in front of her. When she starts reading about (2), there is a bit of suspense before she turns the page and sees what it's all about. Perhaps this little bit of suspense is desirable, but maybe it only annoys her. As she reads about (3) and (4), she has to look ahead at the next page, but that is not so bad because the book is bound so that pages two and three face each other. (The division between two facing book pages is called the 'gutter', so a reference that crosses such a boundary might be known as a 'gutter reference'.) When she gets to (5), though, she has to turn the page to find the picture, and there is a good deal of text to come back and read after she has looked at picture 5. This time it isn't even good for suspense, it is just annoying.

The placement of the illustrations in figure 1 follows a simple rule: when a citation is encountered, put the illustration on the next page that has room for it, but not before any earlier illustration. This is the kind of rule used by most typesetting systems that allow 'floating' insertions. Such a feature is handy to have, but,—as in this example—there are times when a good deal of improvement can be made.

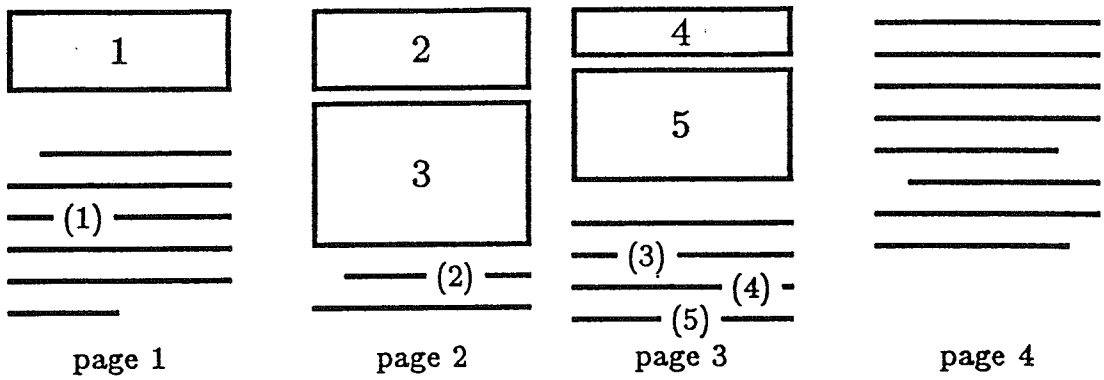


Figure 2. Another way.

Figure 2 shows a better way of paginating this little book. The problem with picture 2 has been alleviated by moving a line from page 1 to page 2, leaving a little more white space on page 1. By moving picture 3 to page 2, enough space has been freed on page 3 for picture 5, thus solving the other problem handily. Note that these two changes do not interfere with each other; but if citation (5) had occurred one line later, they could not both be made. This illustrates that the way early page boundaries are chosen can influence how later breaks are made, and vice versa. To do a really good job, the pagination routine should therefore consider the book (or chapter) as a whole.

PRACTICES OF EARLY PRINTERS

The art of printing is about 500 years older than the use of computers, so it is appropriate to take at least a brief look at how compositors of the past have dealt with pagination. The earliest printers did not publish descriptions of the methods of their craft, preferring to pass this information down by way of tradition and apprenticeship. Therefore what we know about their methods has been deduced primarily by examining their work; fortunately, the quality of the paper and ink they used was sufficiently good that some of their products have lasted into our own time.

According to an account by Douglas McMurtrie [1], the first illustrated books were printed by Albrecht Pfister starting in 1460, about 15 years after movable types had come into practical use. The illustrations were done by using woodcuts, and at first they were all full page illustrations printed on a separate press run from the text. Not until 1472 did a book appear with the text and illustrations printed in the same impression; this technique

was first mastered by Günther Zainer. The use of woodcuts continued to increase and develop until the last part of the eighteenth century, although in the later years the ballooning volume brought a corresponding decrease in quality. The introduction of lithography brought to an end the widespread use of woodcuts for book illustration. We can only guess about how the early printers undertook the task of fitting the copy in with the figures; it is clear that, with the labor involved in making the woodcuts and hand-setting the type, any time used in the planning of the book was time well spent.

MOXON'S ACCOUNT

The first printer to describe in print the details of his craft was Joseph Moxon, whose *Mechanick Exercises* on printing began to come out in parts in 1683 [2]. He described the techniques of hand composition in such detail that his words were copied by later authors of printer's handbooks. Thus, although there are only about fifty extant copies of the edition printed by Moxon, and although the only reprint prior to 1958 was a limited edition, the work was of great value to printers up until the end of the nineteenth century, when the Linotype and Monotype machines were introduced. Moxon's book remains to be of interest to those who, for artistic or recreational purposes, still use hand composition.

Moxon does not say a great deal about pagination, probably relying on the common sense of the compositor to solve any tricky problems as they were encountered. He does have this to say concerning widow lines:

> When in *Composing* he comes near a *Break*, he for some *Lines* before he comes to it considers whether that *Break* will end with some reasonable *White*; If he finds it will, he is pleas'd, but if he finds he shall have but a little single *Word* in his *Break*, he either *Sets* wide to drive a Word or two more into the *Break-line*, or else he *Sets* close to get in that little Word, because a *Line* with only a little Word in it, shews almost like a *White-line*, which unless it be properly plac'd, is not pleasing to a curious Eye.
>
> Nor do good *Compositors* account it good Workmanship to begin a *Page* with a *Break-line*, unless it be a very short *Break* and cannot be gotten in the foregoing *Page*; but if it be a long *Break*, he will let it be the *Direction-line* of the fore-going *Page*, and *Set* his Direction at the end of it.

The meaning of this passage is not readily apparent unless we know the terminology used. A *break* is the white space used to fill out a short line, such as the last line of a paragraph. In the first paragraph Moxon tells us

that a compositor should avoid setting only a single short word at the end of a paragraph, even though to do this it is necessary to look ahead. To understand the meaning of the second paragraph, it is necessary to know that it was the custom in Moxon's time to typeset, in the lower right hand corner of each page, the first word or syllable of the next page. This word is called the *direction*, and the line it is set in is called the *direction-line*. So Moxon's remedy for the widow line was to put it in the normally empty space before the direction on the previous page, provided that it fit.

Notice that Moxon says nothing about looking ahead to avoid the widow, in a fashion analogous to the look-ahead for line breaking. Perhaps this was because it would have been too hard for the compositor to look ahead that far, or maybe the availability of the easy solution of filling up the direction-line made a more complicated method unnecessary.

In Moxon's time, each page was made up as soon as the compositor had set enough lines to fill a page. This made revision of the typeset text into a time consuming and error-prone process, as it was sometimes necessary to unlock several pages of type in order to move lines between them, if the revision resulted in more or fewer lines than the original text. Moxon gives a detailed account of this correction process.

ORIGIN OF THE GALLEY

Early in the nineteenth century, the practice of setting the type in long galleys originated in newspaper offices. The type was printed, proofread, and corrected in this form before it was finally broken into pages for imposition and printing. This made the correction of the text much easier, and allowed a division of labor between the person who composed the lines and the one who made up the pages. This division of labor became more pronounced after the introduction of the Linotype and the Monotype—these machines made the process of typesetting the lines into a keyboard activity. But neither of these machines could assemble type into pages, nor could they handle the complexities of mathematical formulas. The building up of these formulas, addition of diagrams, separation into pages, insertion of displayed lines and pages, and the locking up of type into pages still had to be done by hand. The compositor responsible for these tasks was called the 'maker-up' [3,4].

THE AUTHOR'S RESPONSIBILITY

The job of the maker-up is one that he often cannot perform satisfactorily without the help of the author. T. L. DeVinne [5] makes this clear when he exhorts the author to read the proof, and warns that he will surely find "subheadings, footnotes, extracts, tables, and illustrations contrary to the plan of the copy and in unexpected positions." The proofreader and the

maker-up (and, we might add, the automatic typesetting system) cannot rearrange the composition without instructions from the author. The author alone has the authority and responsibility to insert or delete lines around a table or illustration that may prevent proper make-up.

FOOTNOTES

DeVinne also has something to say about footnotes, calling them a "hindrance in composition and making-up." Most troublesome is a long note that is cited on the last line of the page—this must be split up and continued on successive pages. He gives an example of a particularly trying use of footnotes, printed in 1740. The sample page has but five lines of main text, and some seventy-four lines of footnotes, set in two columns. These notes are also annotated, and the footnotes to the footnotes occupy twelve more lines of six-point type. DeVinne tells us that this is not atypical of the book, but that this page was selected as an example because it was one of the few that could be intelligibly reproduced.

Footnotes must have been in disfavor at the time Moxon wrote, for he makes no mention of them in his work. Neither does he say anything about the making-up of tables and illustrations, although engravings do appear in his book. These engravings are plates that each occupy a full page, and so do not pose major problems to the maker-up. Moxon does describe how to compose side-notes (both cut-in and marginal), so this must have been his preferred method of annotation.

LEADING BETWEEN THE LINES ACCORDING TO ROGERS

The spacing between lines, called the *leading* by typographers, is an important element in the appearance of the printed page. The leading between normal lines of text should generally not be varied, once a suitable value has been decided upon; the leading between larger units is subject to more variations, depending upon the material being typeset. Variations in the leading have an obvious impact on pagination, and it is important to decide how much variation is allowable in what places before asking a machine to decide how to break the material into pages. This decision is largely a matter of taste. Some people have devoted much time to developing their taste in such matters, and we ought to pay attention to the conclusions they have reached, even if we do not always choose to follow their advice. Bruce Rogers, a book designer, has this to say [6]:

> Uneven leading or extra leading between paragraphs may sometimes be necessary in a reference or other special kind of book, but for ordinary text it throws lines out of register, interrupts the continuity of the text, and offends the eye. It

> might therefore be said that the leading of a book should be uniform throughout, with no extra space between paragraphs.
>
> In books of maxims or short extracts several leads between the selections are desirable where considerations of length will not permit full blank lines, and where the extracts are not numbered or titled or dated. In setting such matter it is also advisable to set the first lines without indention. This will serve still further to indicate the fragmentary character of the text, and to distinguish the beginning of each excerpt from any paragraphs that may occur within it, which should of course be given the usual indention.
>
> It is sometimes unavoidable to make facing pages either long or short to take care of 'widow' lines; but every effort should be made to avoid them if the make-up can be rearranged without too much overrunning in order to gain or lose lines. It seems a little too finicky to demand consistent uniformity in length of page throughout a book, especially when some pages may run short for textual reasons. There is entirely too much stress nowadays put on uniformity in composition. A few obvious variations frequently help the appearance.

Sometimes even the leading between normal lines of text needs to be altered slightly; Rogers describes a problem that arose in the design of his version of Homer's *Odyssey*, in which Book XXII began with two figures that had to be printed on facing pages:

> Unfortunately the preceding Book ended with only a few lines on the left-hand page, leaving the opposite a full blank. So with the addition of thin cards we reduced the lines per page from thirty-one to thirty throughout Book XXI, thereby gaining enough lines to fill the short page and carry over six lines to the blank preceding Book XXII. I have never heard of anyone's noticing this discrepancy in Book XXI. It was only one of the tricks of the trade, resorted to by many early printers.

One last bit of advice from Rogers:

> Don't try to 'design' every page of type throughout a book, or work it over too carefully after the style is chosen; leave something to accident, so long as it is not a glaring defect.

THE BADNESS FUNCTION

To program a computer to produce 'readable' books, we must formally define what 'readable' means. One approach is to define a mathematical function, called the *badness function*, that depends on the way the text is broken into pages. This function ideally has the property that the less readable the book is, the greater the value of the function will be. Of course, such a function cannot hope to capture all the nuances of readability; it will ignore such things as the sentence structure, the design of the typefaces, the different reading habits of the various readers, and so forth. Such things fall outside of the scope of the pagination algorithm; they are the responsibility of the author, the typographer, the editor, and the book designer. The badness function may depend on the distribution of white space, the placement of illustrations relative to their citations in the text, and whether the page breaks come in logically desirable places.

Once a badness function has been defined, we need an algorithm that will try to minimize its value. While it is certainly valid to use an algorithm that may not always find the exact minimum, or one that might take a very long time in rare cases, we shall concentrate our attention on algorithms that are guaranteed to find the exact minimum in a reasonable amount of time. For any given badness function, it may or may not be possible to find such an algorithm. Indeed, chapter two demonstrates some badness functions for which there is good reason to believe that no good minimization algorithm exists. Fortunately, as shown in chapters three and four, there are also badness functions that model the intuitive notion of badness fairly well, and that also have a reasonably fast minimization algorithm.

SOURCES OF FLEXIBILITY IN THE TEXT

To take full advantage of the badness function approach, there must be some flexibility in the text. Suppose, for example, that a book design called for each line to take exactly one unit, and each figure to take an integral number of units, where there is a fixed number of units per page, and where each figure is required to appear on the same page as its citation. Then there would either be a unique way to break the text into pages, which could be found by a simple algorithm, or there would be no way at all to do it according to this design. But in general, the pagination algorithm is allowed a little more leeway in its treatment of the text.

The example at the beginning of this chapter demonstrates two ways of exploiting flexibility in the typographic conventions to improve the pagination. One is the movement of the figures in relation to where they are cited

in the text. The other is manipulation of white space, in a way analogous to the way the size of interword spaces are increased or decreased to justify a line. Within a paragraph, there is little of this kind of flexibility, since the eye can notice very small changes in the spacing between lines. However, the space between paragraphs may often be expanded by a small fraction of the baseline spacing without offending the critical eye. The largest amount of white-space flexibility will come from the white space around displays, figures, and at chapter and section heads.

These are some things a pagination algorithm might vary:

a) Placement of the figures in relation to the text
b) White space around displays
c) White space between paragraphs
d) Size of the figures
e) White space between lines
f) Depth of the pages
g) Horizontal spacing in paragraphs, to alter the number of lines
h) Width of the pages
i) Size of the type
j) Order of the figures.

These possibilities are listed roughly in the order of decreasing usefulness to an automated approach. The approach we will consider allows (a) thru (f). The possibility of allowing the pagination routine to influence the way lines are broken, as in (g), is discussed in chapter four. The width of the pages and size of the type are parameters of the book design, and probably should not be altered at the whim of the pagination routine. Similarly, the order of the figures should conform to the plan of the author—composing them out of order would only lead to confusion on the part of the reader.

SCOPE OF RESULTS

The composition of a page is inherently a two-dimensional problem. Illustrations might occupy only a fraction of the full measure, allowing two or more of them to occupy the same vertical height. Or the text may be run around the figure; if this is done before pagination is performed, the place that the figure occupies cannot be broken across a page boundary. But if the figure could be moved up or down a few lines, the pagination could probably be improved. The use of multiple columns introduces further complications, especially if partial columns at the end of each section must be made to balance. Some further problems of page make-up, and the implications for

automatic typesetting systems, are described in a brief article by Paul Justus, 'There is more to typesetting than setting type' [7]. It is a challenge to write a system that can cope with the combination of these make-up conventions well enough to produce a reasonably well-formatted document. It is even harder to design an optimizing make-up procedure, even assuming the existence of a badness function that can take into account the many trade-offs involved. As we are interested here primarily in exact minimization procedures, we will restrict ourselves to a one-dimensional model. Furthermore, we will not concern ourselves with how the typographical components are arranged on the page, but merely with what goes on each page.

THE MODEL: BOXES, GLUE, PENALTIES, AND INSERTS

The model we shall use was developed by D. E. Knuth for the TEX typesetting system [8]. In this model, the components of the page are pictured as being enclosed in *boxes*. Boxes are rectangles with sides that are parallel to the edges of the paper. At the lowest level, each character is a box whose height and width correspond to the height and width of the character. The height of a character is measured from the baseline, which is the lowest point of the letter 'A'. Just as some letters ('p', for example) extend below the baseline, so may boxes. The amount of this extension is called the *depth* of the box. A row of boxes, called a *horizontal list* in TEX terminology, may be stuck together and enclosed in a box to form a line. To keep the words from all running together, they are separated by *glue*. Glue has properties that resemble a glob of silicone sealant, or a small metal spring. Like a box, a piece of glue has a normal width. It also has two other parameters, the *stretch* and the *shrink*. These tell TEX by how much the glue can be expanded or compressed. For example, the space between words normally has a width of 6 units, a stretch component of 3 units, and a shrink component of 2 units, where the size of a unit depends on the current font. When a horizontal list is boxed to a certain size, for example to make it fit the measure of the page, all the pieces of glue in the list are expanded or compressed in proportion to their stretch or shrink so that the resulting box will be exactly the desired size. Once this is done, the glue is 'set' and does not change its size any more, the enclosing box being treated as a unit. The height of the box around a horizontal list is calculated as the maximum of the heights of the component boxes, and similarly for the depth.

Just as boxes can be combined in a horizontal list, they may be combined in a vertical list. The glue between items in a vertical list works in the same

way as horizontal glue, but it usually arises in a different way. Whereas the horizontal glue usually comes from a space between words, the vertical glue is normally calculated so that the distance between baselines is a fixed value specified by the user, unless this is impossible due to an unusually high or deep box. The user may also specify either kind of glue explicitly.

It is possible for the user to specify exactly what goes into each box. More commonly, TEX will take a long list of boxes and break it up into several lists, each of which is boxed separately. If the original list was a horizontal list, these new boxes form lines, and get joined together in a vertical list. If the original list was a vertical list, then each new box forms a page, which gets sent to the output file after the addition of such things as page numbers and running heads.

BREAKING PARAGRAPHS INTO LINES

To break a horizontal list into lines, TEX uses a dynamic programming algorithm to decide on the 'best' way to do the breaking. This algorithm, analogous to those for page breaking that will be developed in chapters three and four, finds a way to break the paragraph into lines so that a certain badness function is minimized. One component of the badness function measures the amount that the glue has to be stretched or compressed in order to make a line fit. If the normal spacing of the line is short of the measure by the amount x, and if y is the total amount of stretchability in the line, then the *adjustment ratio* r is x/y; if the normal spacing of the line is wider than the measure by the amount x and the total shrinkability in the line is z, then r is $-x/z$.

TEX is not allowed to break a list just anywhere; most commonly a break is made between a box and some glue that immediately follows it. The only other place that a break may occur is at a *penalty node*. A penalty node's purpose is to control the breaking of the list. It has an associated penalty value, which is added to the total badness if a break is made at the penalty node. This value may be positive, to discourage a break, or negative, to encourage one. By convention, any value 1000 or greater is treated as infinity, so '*penalty* 1000' means 'never break here,' and '*penalty* –1000' means 'always break here.' When a break is chosen, all of the glue and penalties that follow it and precede the next box in the list are discarded before the new boxes are made.

TEX calculates the badness function of a paragraph by figuring out how many *demerits* to charge for each line, and then adding together all the demerits. The demerits for a line having adjustment ratio r and ending with

a penalty node with the value P is calculated according to the formula

$$(1 + 100|r|^3 + P)^2,$$

provided that $r \geq -1$ and $P \geq 0$. If $r < -1$, the badness is taken to be infinite, since TEX does not allow glue to be compressed by more than its shrinkability. If $P < 0$, the demerits are calculated as

$$(1 + 100|r|^3)^2 - P^2,$$

in order to keep the function monotone when the penalty is negative. Additional demerits are charged when both this line and the one before it end in a hyphen. This particular definition is quite arbitrary, but it works out quite well in practice, as you can see by looking at this page.

ADVANCED USES OF PENALTIES AND GLUE

When the boxes-and-glue model was first proposed, it was apparent that it would allow many of the common typesetting operations to be done without introducing new primitives for each. Justification is an obvious example, accomplished by using glue with positive stretchability, shrinkability, and width for each blank, and by suppressing the glue at the ends of the lines. The glue after punctuation marks may have more stretch and less shrink than a normal space. By putting glue with a very large stretchability at the right end, at the left end, or at both ends of a line, the line can be left justified, right justified, or centered. On a title page, it is often desirable to 'center' some text on the page, but what is wanted is not the exact center, but the *visual center*, which is a bit higher on the page. This is easily done by putting some glue with large stretchability above the text, and some glue with an even larger stretchability after the text.

RAGGED LEFT MARGINS

After TEX had been implemented and in use for a while, it was discovered that by using boxes and glue in conjunction with penalty nodes and the optimizing line breaking algorithm, it was possible to do other interesting things without introducing new primitives. For instance, to set a paragraph with a ragged left margin and a straight right margin, the user may specify

penalty 0
glue 4*pt plus* −1000*pt*
box width 0*pt*
penalty 1000
glue 0*pt plus* 1000*pt*

for each space in the text. The only place a break may occur is at the *penalty* 0 node; if no break occurs here, the negative stretch in the first glue cancels out the stretch in the second glue, and the net effect is 4 points of fixed-width glue. If a break does occur, the first glue is discarded, and the empty box causes the second glue to be retained, so this glue appears at the front of the next line. To get the boundary conditions right, the paragraph should begin with '*glue* 0*pt plus* 1000*pt*' and it should have no extra glue at the end. (TEX normally adds glue with 100000 points of stretch at the end of each paragraph to set the last line flush left.)

By means of analogous sequences, it is possible to get the line breaking algorithm to set a paragraph with a ragged right margin, or even to center each line of the paragraph. These and other similar sequences may be discovered with an algebra of boxes and glue that was developed by D. E. Knuth [9]. Of course, similar techniques may be used in vertical mode to allow, for instance, the amount of white space at the bottom of the page to vary in order to give the pagination routine some extra flexibility.

INSERTS

In vertical mode, the box-and-glue model is not quite sufficient to do automatically everything that is needed for practical typesetting. There must be a way of specifying figures and tables that are allowed to 'float' with respect to the text; their position in the final document is not specified exactly by the author, but determined by constraints such as 'this figure should appear at the top of a page somewhere nearby' or 'this footnote must appear at the bottom of the present page.' In TEX, such things are specified as *inserts*. An insert is a vertical list that may appear in the document at some place other than where it is defined. Since an insert contains a vertical list, it has a normal height, stretch, and shrink, just like glue. When the insert is specified, the user tells whether it is to appear at the top or bottom of the page (i.e., whether it is a *topinsert* or *botinsert*), and whether it must appear on the same page as the definition (*bound*), or may appear on a different page, if necessary (*floating*). The floating topinsert is most useful for inserting figures or tables, since these most often come at the top of a page. The bound botinsert is useful for short footnotes that never have to be broken across a page. It is possible to think of other varieties of inserts that would be handy, such as 'put this algorithm definition right here if it will fit; otherwise put it at the top of a facing page,' or 'if this is the first figure on the present page, put it at the top; otherwise put it at the bottom.' Since such variations deal with how things are arranged on the page rather than

with what the page is to contain, they do not affect the pagination algorithm unless the different arrangements take up different amounts of space.

CALCULATION OF INTERLINE GLUE TO LINE UP BASELINES

Books are normally printed so that the top baselines on all the pages line up, and similarly for the baselines of the last line on each page. To allow for this convention, TEX adds some glue before the first box on each page with a size calculated to put the top baseline a fixed distance, specified by the user, from the top of the page. It may happen that the height of the first box on the page is greater than this distance; should this happen, TEX does not add any such glue. Similarly, the glue in the vertical list is set so that the baseline of the last line of the page coincides with the bottom boundary of the page. If this depth is greater than a specified maximum, as may happen with a built-up box, the baseline of the bottom box is effectively lowered so that the depth does not exceed the maximum.

This convention of aligning the bottom baselines on the page makes the pagination routine work a little harder; if the depth of a botinsert differs from the depth of the last line of text on the page, the nominal height of the material on the page will change when the insert is moved to the bottom of the page. This needs to be taken into account when doing the pagination; we shall see in chapter four that, although it adds a little complication to the code, it is not a major obstacle.

DROPPING OR ADDING LINES AT THE ENDS OF PAGES

Sometimes a book designer wishes to allow the bottom margin of the page to vary by a line or two in order to avoid widow lines; one way to do this is to specify a sequence of penalty and glue items analogous to that used for a ragged right margin. But it is often desirable in a case like this to make the distance between the last baseline on the page and the normal bottom baseline position a whole number of baseline units, so that the ending lines will align with the text on the surrounding pages. This can be accomplished by inserting the following sequence between the lines:

penalty 0
glue 2*bu* — (depth of previous box)
penalty 200
glue —1*bu*
penalty 100
glue —2*bu*
penalty 300
glue 2*bu* — (height of next box)

where '*bu*' stands for the nominal distance between baselines. This sequence allows the page to be short by one or two lines, or long by one line, and charges penalties of 100, 200, and 300, respectively.

TREATMENT OF FOOTNOTES

Some styles allow long footnotes to be broken up and continued on the next page. Although such things are to be avoided wherever possible because they make it harder for the reader to follow the footnotes, TEX's model is capable of supporting them in the following way: make each line of the multi-line footnote into a bound botinsert, and separate each of them by a penalty node with a positive penalty. Since there is no penalty or glue between the first line of footnote and the preceding box, no break can occur there, and at least the first line of the footnote must come on the same page as the citation. The penalty nodes will allow the footnote to be broken if necessary, but these breaks will be avoided where possible because of the positive penalty values. If appropriate ordering of the inserts on the bottom of the page were provided, footnotes-within-footnotes could also be supported by this method, although this style should not be encouraged. It should be noted here that the current implementation of TEX does not automatically break up long footnotes in this way, and to do the specification manually would probably be more work than just using one botinsert for the footnote, and breaking it in two if trouble arises.

The pagination algorithm can handle bound inserts with little more trouble than just boxes, glue, and penalties. For the purposes of pagination, a bound insert may almost be treated as the sequence

penalty 1000
glue (normal size) *plus* (stretch) *minus* (shrink)
box height 0

where the penalty prevents a break from occurring at the glue, and where the null box allows a break to happen after the insert, should it be followed by glue. The normal size, stretch, and shrink of the glue are equal to the corresponding components of the insert. After the pagination routine has used this substitution to decide on where the page breaks should be, the paste-up routine can put the insert at its proper position on the page. We say this substitution is 'almost' sufficient because the pagination routine also has to remember the depth of the last botinsert on the page, and use this instead of the depth of the last text line when calculating what the depth of the whole page is going to be.

Chapter 2

Some NP-complete Pagination Problems

Breaking up is hard to do. — N. Sedaka

An optimist might hope to find a pagination algorithm that would take an arbitrary computable badness function and some text, and find the way of dividing the text into pages that achieves the smallest possible value of the badness function. One way to do this is to by trying all ways of paginating the document. It is not very hard to count how many different ways there are to do this; if there are n lines of text and m figures, and if there must be at least one line of text on each of p pages, then $p-1$ of the $n-1$ spaces between the lines will be chosen as page boundaries. After this has been done, m of the p pages will be chosen, with repetition allowed, to hold the figures. By the techniques of elementary combinatorics, the number of ways to break is therefore

$$\binom{n-1}{p-1}\binom{m+p-1}{m}.$$

If we make the assumption that both m and n grow in proportion to p, say $m = \mu p$ and $n = \nu p$, then we find by Stirling's approximation that

$$\binom{\nu p - 1}{p - 1}\binom{\mu p + p - 1}{\mu p} = \Theta(c^p/p) \quad \text{as } p \to \infty,$$

$$\text{where } c = \frac{\nu^\nu}{(\nu-1)^{\nu-1}} \frac{(\mu+1)^{\mu+1}}{\mu^\mu} > 1,$$

provided that $\nu > 1$ and $\mu > 0$. Thus the number of possibilities grows exponentially as the size of the problem increases. Since a typical value for ν is around 50, p does not have to get very large before the number of possibilities becomes unmanageable.

A brute-force approach that examines all possibilities is the best we can do if we are not allowed to use any special properties of the badness function. This may be shown by a simple adversary argument: suppose we were given a purported algorithm that didn't need to look at all of the possibilities. We run it with a particular positive-valued badness function and some text as input, and note the places at which the badness function is evaluated. Then we choose one of the places at which the badness function was not evaluated, and define a new badness function that is the same as the old one, except that it takes the value zero at this one place. Now if this new badness function and the same text are given to the algorithm, it will find the same answer as before, which is not the right answer for the modified badness function.

From this argument we see that a good pagination algorithm will work only for a restricted class of badness functions. Since most of the ways of paginating a document will have a very high badness according to any 'reasonable' badness function, our optimist might hope to find a large class of such functions that can be handled by an efficient pagination algorithm. But this hope, too, will be crushed by the results presented in this chapter. We will examine some badness functions that seem to be quite reasonable on the surface, but whose minimization problems are NP-complete.

WHAT EFFICIENT MEANS

For the purposes of this chapter and the next, we shall call an algorithm *efficient* if its running time is bounded by some polynomial in the size of the input. This is a useful definition because the class of such efficient algorithms is invariant over a wide variety of machine models, including deterministic Turing machines, pointer machines, and random-access machines.

The *nondeterministic* Turing machine is an imaginary machine that is like no machine anyone has ever seen. It is allowed to make guesses, and take branches based on those guesses. Given a sequence of guesses, the machine may halt after some number of steps. The running time of the nondeterministic Turing machine is the minimum number of steps executed for any sequence of guesses. The set NP is the set of all problems that can be solved in polynomial time on a nondeterministic Turing machine.

The theory of NP-complete problems was established in the late 1960s when S. Cook [10] showed that the satisfiability problem for propositional calculus is at least as hard as any problem in NP, in the sense that if there existed an efficient algorithm for the satisfiability problem, then any problem in NP would be solvable by an efficient algorithm. A problem that shares this property with satisfiability is called *NP-hard*, and if it is also in NP, it is called *NP-complete*. Since no efficient way of solving the satisfiability problem, or any NP-complete problem, has ever been found, an NP-complete problem is normally considered to be intractible. In 1972, R. Karp [11] showed several important problems to be NP-complete, and since then hundreds of other problems have been shown to belong to this class, many of them of practical importance. A collection of most of these results to date, as well as a lucid introduction to the theory of NP-completeness, may be found in the book by M. Garey and D. Johnson, *Computers and Intractability* [12].

It is convenient to restrict the discussion to decision problems when proving NP-completeness results. A decision problem is one that asks a yes-no question about the input. Some problems are already decision problems. For example, the satisfiability problem takes a formula in the propositional calculus and asks whether or not there exists an assignment of truth values to the variables such that the formula is true. A minimization problem, such as the pagination problem, may be converted to a decision problem by including as part of the input a bound B, and asking if there is a way to give the variable to be minimized a value of at most B.

For the purposes of proving NP-completeness results, it is usually best to specialize the general problem so that irrelevant details are suppressed. Each of the proofs in this chapter will need a slightly different specialization, so the correspondence to the general model will be explained for each proof when it is not readily apparent.

QUADRATIC BADNESS FUNCTION

The first NP-completeness result uses a badness function for the figure placement that is the sum of the squares of the differences between the page numbers of the citations and the page numbers of their corresponding figures. Each figure in this problem may have more than one citation that refers to it, and in fact there may be many references from a given text block to a given figure. We will denote the number of references between text block t_i and figure f_j by the function $W(t_i, f_j)$. In order to obtain a strong NP-completeness result, we will bound W by a polynomial q in the size of the problem. The same polynomial applies to all instances of the problem; we will choose q later so that all the instances we need will satisfy this constraint.

Page Breaking Problem with Multiply Referenced Figures and Quadratic Badness Function (MQ):

GIVEN: A set $T = \{t_1, t_2, \ldots, t_N\}$ of text blocks,
a set $F = \{f_1, f_2, \ldots, f_N\}$ of figures, with $T \cap F = \emptyset$,
a mapping $W : T \times F \to \{0, 1, 2, \ldots, q(N)\}$, and a bound B.

QUESTION: Is there a 1-1 mapping $P : (T \cup F) \to \{1, 2, \ldots, 2N\}$
such that $P(t_i) < P(t_j)$, $P(f_i) < P(f_j)$ for $1 \leq i < j \leq N$,
and

$$S = \sum_{1 \leq i,j \leq N} W(t_i, f_j)(P(t_i) - P(f_j))^2 \leq B?$$

Before we become involved in the details of the construction, let us take a few moments to examine how this version of the problem is related to real-life page breaking problems. Suppose that the text we are typesetting is not allowed to be stretched or compressed to fit on a page and that each of the figures exactly fills one page. Then the page boundaries must always occur at the same places in the text, no matter where the figures are placed. Each page of text material t_i may have any number (up to a limit imposed by the page size) of references to each figure f_j; the function $W(t_i, f_j)$ is just the number of such references. The function W is bounded by the polynomial q because we cannot realistically expect there to be exponentially many references in the document. Of course, it may be that the user of the typesetting system is allowed to specify a weight for each reference; in such a case an arbitrary function W would not be unrealistic. The function P just specifies the order in which we choose to place the text and figures. The particular badness

function we have chosen contributes a quantity proportional to the square of the distance (within the linear embedding) between a figure and a reference to it. This is in some sense the simplest function for which the following proof technique will work. We shall see in the next chapter that the problem has a good solution if the square in the badness function is replaced by the absolute value.

Now that we have a feeling for why this problem is relevant, let us proceed with the proof of its NP-completeness.

Theorem 1. MQ is NP-complete.

Proof: The problem is evidently in NP, since a nondeterministic Turing machine can guess the embedding P, and test it to determine whether or not it satisfies the required properties, all in polynomial time.

To finish the proof, we will show how to transform an instance of Maximum 2-Satisfiability into an instance of MQ.

The Maximum 2-Satisfiability problem is defined as follows:

MAX 2-SAT:

GIVEN: A set of variables $X = \{x_1, x_2, \ldots, x_n\}$;
a set of clauses $C = \{u_1 \vee v_1, u_2 \vee v_2, \ldots, u_m \vee v_m\}$,
where $u_i, v_i \in \{x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_n, \bar{x}_n\}$;
and a bound $K \leq |C|$.

QUESTION: Is there an assignment of truth values to the variables x_i such that at least K of the clauses are satisfied?

MAX 2-SAT was shown to be NP-complete by Garey, Johnson, and Stockmeyer [13] by a transformation from 3-satisfiability.

Given an instance of MAX 2-SAT, we will describe a transformation to an instance of MQ that has the same answer.

Let $N = 2n + 2m$, $T = \{t_1, \ldots, t_{2n+2m}\}$, and $F = \{f_1, \ldots, f_{2n+2m}\}$. The weight function W is constructed as follows: First set $W(t_i, f_i) = b_1$ for $1 \leq i \leq N$, where b_1 will be chosen large enough to insure that the function P must satisfy $|P(t_i) - P(f_i)| = 1$. Thus the only choices left for the function P will be whether to put t_i or f_i first, for each i. The first $4n$ positions of the embedding will be used to indicate the truth values of the

variables x_i. The occurrence of

$P(k)$:	$4i-3$	$4i-2$	$4i-1$	$4i$	
k:	t_{2i-1}	f_{2i-1}	f_{2i}	t_{2i}	corresponds to x_i being true, and
k:	f_{2i-1}	t_{2i-1}	t_{2i}	f_{2i}	corresponds to x_i being false.

To guarantee that the embedding conforms to one of the above patterns, let $W(t_{2i-1}, f_{2i}) = W(t_{2i}, f_{2i-1}) = b_2 \quad (1 \leq i \leq n)$, where a suitable value for b_2 will be given later. Note that these cause a contribution to the total badness in the amount of $8b_2$ for each of the allowed patterns above, but $10b_2$ for the alternative patterns $t\,f\,t\,f$ and $f\,t\,f\,t$. These pairs of weight function values will be subsequently referred to as *communication pairs*, since they communicate the embedding between two t-f pairs.

The remaining $4m$ positions of the embedding will correspond to the clauses. The correspondences are as follows:

$P(k)$:	$4n+4j-3$	$4n+4j-2$	$4n+4j-1$	$4n+4j$		
k:	$t_{2n+2j-1}$	$f_{2n+2j-1}$			$\Leftrightarrow$	u_j true,
k:	$f_{2n+2j-1}$	$t_{2n+2j-1}$			$\Leftrightarrow$	u_j false,
k:			t_{2n+2j}	f_{2n+2j}	$\Leftrightarrow$	v_j true,
k:			f_{2n+2j}	t_{2n+2j}	$\Leftrightarrow$	v_j false.

To link the values of the literals to the values of the corresponding variables, we add the following communication pairs:

$$\left.\begin{array}{lll} W(t_{2n+2j-1}, f_{2i}) & = W(t_{2i}, f_{2n+2j-1}) & = b_2 \text{ if } u_j = x_i \\ W(t_{2n+2j-1}, f_{2i-1}) & = W(t_{2i-1}, f_{2n+2j-1}) & = b_2 \text{ if } u_j = \bar{x}_i \\ W(t_{2n+2j}, f_{2i}) & = W(t_{2i}, f_{2n+2j}) & = b_2 \text{ if } v_j = x_i \\ W(t_{2n+2j}, f_{2i-1}) & = W(t_{2i-1}, f_{2n+2j}) & = b_2 \text{ if } v_j = \bar{x}_i \end{array}\right\} \; 1 \leq j \leq m$$

Each of these weights will contribute $2d^2b_2$ to S if the truth setting of the literal corresponds to the truth setting of its variable, where d is twice the difference between the subscripts of the two arguments to the weight function. For instance, if $v_j = \bar{x}_i$, then $d = 2(2n + 2j - (2i - 1))$. If the truth values do not properly correspond, the contribution will be

$$((d-1)^2 + (d+1)^2)b_2 = 2(d^2+1)b_2.$$

The remaining values of W will encode the semantics of the clauses. For $1 \leq j \leq m$, let

$$W(t_{2n+2j-1}, f_{2n+2j}) = 3 \quad \text{and} \quad W(t_{2n+2j}, f_{2n+2j-1}) = 5.$$

These are called the "clause pairs."

Then the four possible truth assignments in a clause make the following contributions to the badness:

u true, v true	$\Leftrightarrow$	$t\ f\ t\ f$	contributes $3^2 \cdot 3 + 1^2 \cdot 5 = 32$
u true, v false	$\Leftrightarrow$	$t\ f\ f\ t$	contributes $2^2 \cdot 3 + 2^2 \cdot 5 = 32$
u false, v true	$\Leftrightarrow$	$f\ t\ t\ f$	contributes $2^2 \cdot 3 + 2^2 \cdot 5 = 32$
u false, v false	$\Leftrightarrow$	$f\ t\ f\ t$	contributes $1^2 \cdot 3 + 3^2 \cdot 5 = 48$.

Let W be zero for all values in its domain for which it has not yet been defined.

Now let us fix the bound $B = B_1 + B_2 + B_3$ so that the embedding P is forced to have all the required properties.

Let $B_3 = 48(m - K) + 32K$, which allows enough to embed K or more of the clause pairs in their satisfied configuration, leaving the rest unsatisfied.

Let $B_2 = 4b_2 \sum (i - j)^2$, where the sum is over all pairs (i, j) such that $W(t_i, f_j) = b_2$. This permits the communication pairs to be embedded properly.

Let $b_2 = 8(m - k) + 5$. It is proved below that if any of the communication pairs are embedded improperly, this value of b_2 causes an increase in badness larger than any accompanying decrease due to more clauses being satisfied.

Let $B_1 = Nb_1$. This permits each t_i to be next to its f_i.

Let $b_1 = \lceil (B_2 + B_1)/3 \rceil + 1$. We will see in a moment that if any t_i is not next to its f_i, this value of b_1 causes an increase in the badness larger than any decrease that could arise from fouling the communication pairs or the clause pairs.

Let $q(i) = 64(i+1)^4$; this makes $q(N) > b_1 \geq b_2 > 5 > 3$, so W will be bounded as required.

Now if we are given a truth value assignment to $x_1, x_2, \ldots, x_n$ that makes K or more clauses true, it is clear from the remarks above how to construct P to satisfy the bound B. Conversely, given a P that satisfies the bound B, it must satisfy the following conditions:

- $|P(t_i) - P(f_i)| = 1$ for $1 \leq i \leq N$, or else $S \geq (N - 1 + 2^2)b_1 > B$;
- the communication pairs are set properly, or else $S \geq B_1 + B_2 + 2b_2 + 32m > B$;
- at least K clause pairs must be embedded in their satisfied position, or else $S > B$.

Thus the function P can be used to construct a truth assignment to the variables $x_1, x_2, \ldots, x_n$ that satisfies at least K of the clauses. This completes the proof of theorem 1.

The MQ problem shows that there is at least one badness function that leads to a hard pagination problem. The essential features of the badness function that the proof relies on are that the function $f(x, y) = (x - y)^2$ has a smaller value when $|x - y| = 1$ than when $|x - y| > 1$ and that $f(a, b) + f(a + 1, b + 1) < f(a, b + 1) + f(a + 1, b)$; similar proofs may be constructed for other badness functions that satisfy these properties. By making the construction a little more complicated, chaining together the clause references to x_i instead of referring them all the way back to f_{2i}, the values of b_1 and b_2 could be considerably reduced.

SINGLY FERENCED FIGURES

It may seem that the fact that multiple references per figure are allowed is a necessary prerequisite for this result, but in fact a similar result may be proved about a pagination problem with only one reference per figure. The idea is to replace each figure in the above construction by a collection of figures, one new figure for each reference to the old figure. For want of a better collective noun, we will call this collection of figures a *flock*. The sizes of the figures in a flock add up to the page size; each text block is one page

high, as before. All of the figures in a given flock must fall on the same page, because no figures can share a page with a text block (there is no room), and the ordering of the figures prevents the flocks from getting mixed up. This new problem may be stated as follows:

Page Breaking Problem with Singly Referenced Figures and Quadratic Badness Function (SQ):

GIVEN: A set $T = \{t_1, t_2, \ldots, t_n\}$ of text blocks;
a set $S = \{s_1, s_2, \ldots, s_m\}$ of figures, with $T \cap S = \emptyset$;
a mapping $r : S \to T$, indicating where each figure is cited;
a page size $c \in \{1, 2, \ldots\}$;
a mapping $h : T \cup S \to \{1, 2, \ldots, c\}$, indicating the height of text and figures;
an integer p, the number of pages, where $\sum_{x \in T \cup S} h(x) = pc$;
and a bound K.

QUESTION: Is there a mapping $Q : (T \cup S) \to \{1, 2, \ldots, p\}$
such that $Q(t_i) \leq Q(t_j)$ for $1 \leq i < j \leq n$,
$Q(s_i) \leq Q(s_j)$ for $1 \leq i < j \leq m$,

$$\sum_{x \in Q^{-1}(l)} h(x) = c \quad \text{for } 1 \leq l \leq p,$$

where $Q^{-1}(l) = \{x \in T \cup S \mid Q(x) = l\}$,
and such that $\sum_{1 \leq i \leq m} (Q(s_i) - Q(r(s_i)))^2 \leq K$.

Theorem 2. SQ is NP-complete.

Proof: Clearly SQ $\in$ NP. The reduction is from MQ.
Given an instance of MQ, let

$$m = \sum_{1 \leq i,j \leq n} W(t_i, f_i);$$
$$S = S_1 \cup S_2 \cup \ldots \cup S_n,$$
where $S_j = \{s_{i,j,k} \mid 1 \leq i \leq n,\ 1 \leq k \leq W(t_i, f_j)\}$,
the elements $s_{i,j,k}$ are sorted first by j and then by i and k, and given the names $s_1, s_2, \ldots, s_m$.

Furthermore, let $r(s_{i,j,k}) = t_i$;

$$c = \max_{1 \le j \le n} |S_j|;$$

$h(t_i) = c$ for $1 \le i \le n$;

$h(s_{i,j,k}) = 1$ for $1 \le k < W(t_i, f_j)$;

$h(s_{i,j,k}) = c - |S_j| + 1$ for $k = W(t_i, f_j)$;

$p = 2n$, and $K = B$.

Now given a solution to the instance of MQ, a solution to the corresponding instance of SQ is obtained by setting

$$Q(t_i) = P(t_i) \text{ for } 1 \le i \le n, \text{ and}$$

$$Q(s_{i,j,k}) = P(f_j) \text{ for } 1 \le i, j \le n,\ 1 \le k \le W(t_i, f_j).$$

Because P is one-to-one, $Q^{-1}(l)$ will either be the singleton $\{t_j\}$, for some j, or else the set S_j, for some j; in either case the sum $\sum_{x \in Q^{-1}(l)} h(x)$ will be equal to c. It is not hard to see that the references coded by the functions r and W are in a one-to-one correspondence that preserves the origin and terminus of each reference, so that

$$\sum_{1 \le i \le m} (Q(s_i) - Q(r(s_i)))^2 = S = B \le K$$

Conversely, given a solution to this instance of SQ, we need to construct a solution to the instance of MQ. The most difficult part of this is to show that if a page contains one figure $s_{i,j,k}$, then it must contain exactly the set S_j. Suppose this is not the case, and let l_0 be minimum such that the condition fails for $s_{l_0} = s_{i_0,j_0,1}$. Let $x_0 = Q(s_{l_0})$, the page that the figure s_{l_0} falls on. We cannot have a text block fall on page x_0, or the bound c on the sum of the heights of items on the page would be exceeded. For the same reason the page cannot contain more items than those in S_{j_0}. The figure s_{l_0} cannot be the first thing on the page, because this would force the page to contain exactly S_{j_0}, again because the sum of the heights must be c, and because the mapping Q satisfies $Q(s_i) \le Q(s_j)$ for $i < j$. Thus the page must contain s_{l_0-1}, and not all of its family S_{j_0-1}, again by the bound c. But this contradicts the requirement that l_0 be minimum.

Having shown that Q maps all the elements in S_j to the same value, we may go on to construct a solution to the instance of MQ. Let $P(t_i) = Q(t_i)$ for $1 \leq i \leq n$, and let $P(f_j) = Q(s_{1,j,1})$ for $1 \leq j \leq n$. Then P will satisfy all the requirements of a solution to MQ, completing the proof of theorem 2.

ZERO-ONE BADNESS FUNCTION

The remaining NP-completeness results of this chapter use a different badness function for the placement of the figures, which we will call the zero-one badness function. The badness due to a reference is zero if the citation and the figure fall on pages that have the same 'handedness,' and one otherwise. In other words, if the figure is on page p and the citation is on page q, the associated badness is $(p - q) \bmod 2$. The total badness is just the sum of the badnesses of all the references. This is probably not a badness function that anyone would ever want to use in a real book, but with it we will be able to show that even this simple badness function cannot be efficiently minimized, even when the figures have one citation each and are constrained to appear in the same order as the citations. This result indicates that the class of functions we will be able to handle efficiently is quite limited.

To introduce the details of the construction in a more easily understood form, we will look at a couple of easier constructions along the way to the final result of this chapter.

The first of these constructions is actually quite simple. There will be one text block or figure per page, as was the case for MQ. In this construction the pages are not all the same size—the size of each page is specified as part of the problem instance. The reduction is from 3-SAT, that is, the satisfiability problem for propositional formulas in clause normal form with three literals per clause. An instance of 3-SAT is defined as follows:

Satisfiability with Three Literals Per Clause (3-SAT):

GIVEN: A set of variables $X = \{x_1, x_2, \ldots, x_n\}$, and a set of clauses $C = \{u_1 \vee v_1 \vee w_1, u_2 \vee v_2 \vee w_2, \ldots, u_m \vee v_m \vee w_m\}$, where $u_i, v_i, w_i \in \{x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_n, \bar{x}_n\}$.

QUESTION: Is there an assignment of truth values to the variables x_i such that all of the clauses in C are satisfied?

The problem we will prove to be NP-complete is the following:

Page Breaking Problem with Zero-one Badness Function and Variable Page Size (VZ):

GIVEN: A set $T = \{t_1, t_2, \ldots, t_N\}$ of text blocks;
a set $F = \{f_1, f_2, \ldots, f_M\}$ of figures, with $T \cap F = \emptyset$;
a mapping $r : F \to T$, indicating where each figure is cited;
a mapping $s : \{1, 2, \ldots, M + N\} \to \{1, 2, \ldots\}$, indicating the size of each page; and a mapping $h : T \cup F \to \{1, 2, \ldots\}$, indicating the height of each text block or figure.

QUESTION: Is there an integer p and a mapping $P : T \cup F \to \{1, 2, \ldots, p\}$
such that $P(t_i) \leq P(t_j)$ for $1 \leq i < j \leq N$;
$P(f_i) \leq P(f_j)$ for $1 \leq i < j \leq M$;

$$\sum_{x \in P^{-1}(i)} = s(i) \quad \text{for} \quad 1 \leq i \leq p,$$

where $P^{-1}(i) = \{x \in T \cup F \mid P(x) = i\}$;
and $P(f_j) \equiv P(r(f_i)) \pmod 2$?

Note that this last condition is equivalent to requiring the total badness to be zero.

The pagination forced by the construction will consist of two parts: the first $2n$ pages will encode the variables, and the last $6m$ pages will encode the clauses. Each variable will be encoded in four pages, starting on an odd page, which may be set with zero badness in one of the two ways shown in figure 1.

Each clause will be encoded by six pages that must contain three figures and three text blocks, and that start on an odd page. The last page of the group will be forced to contain a text block. The position of each of the three figures will encode the setting of the corresponding literal, true if the figure falls on an odd page, and false if it falls on an even page. Since there are three figures and two even pages, at least one of the figures must fall on an odd page, so at least one literal in each clause must be true. An example of this encoding is shown in figure 2.

Having seen the general idea of the construction, we now proceed to prove how it works.

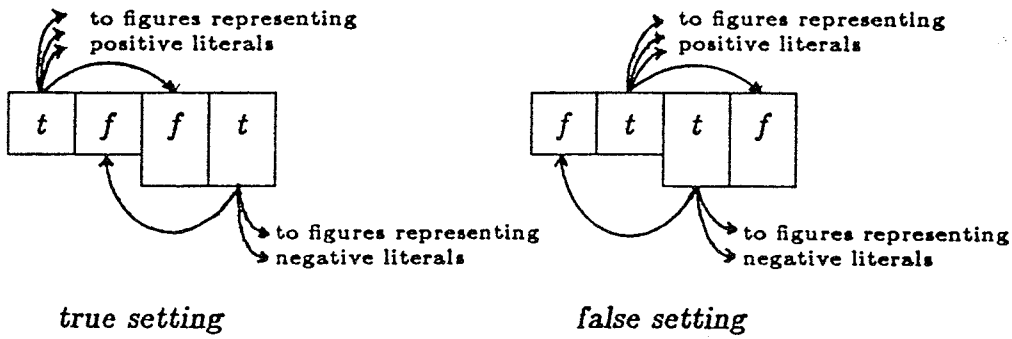
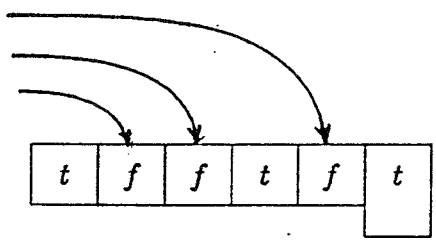


Figure 1. Truth value settings in construction for VZ.



An embedding with the first literal false and the other two true.

Figure 2. Clause encoding in construction for VZ.

Theorem 3. VZ is NP-complete.

Proof: Clearly the problem is in NP. Now given an instance of 3-SAT, we will construct an instance of VZ that is solvable if and only if the instance of 3-SAT is.

Let $N = M = 2n + 3m$;

$h(t_{2i-1}) = h(f_{2i-1}) = 2$ for $1 \leq i \leq n$;
$h(t_{2i}) = h(f_{2i}) = 3$ for $1 \leq i \leq n$;
$h(f_i) = 2$ for $2n < i \leq 2n + 3m$;
$h(t_{2n+3i-1}) = h(t_{2n+3i-2}) = 2$ for $1 \leq i \leq m$;
$h(t_{2n+3i}) = 3$ for $1 \leq i \leq m$.

Furthermore, let $s(4i-2) = s(4i-3) = 2$ for $1 \leq i \leq n$;
$s(4i) = s(4i-1) = 3$ for $1 \leq i \leq n$;
$s(4n+6i) = 3$ for $1 \leq i \leq m$;
$s(4n+6i-k) = 2$ for $1 \leq i \leq m$, $1 \leq k \leq 5$;
$r(f_{2i}) = t_{2i-1}$, $r(f_{2i-1}) = t_{2i}$ for $1 \leq i \leq n$;
$r(f_{2n+3i-2}) = r'(u_i)$, $r(f_{2n+3i-1}) = r'(v_i)$, $r(f_{2n+3i}) = r'(w_i)$
for $1 \leq i \leq m$,
where $r' : \{x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_n, \bar{x}_n\} \to \{t_1, t_2, \ldots, t_{2n}\}$ is defined by $r'(x_i) = t_{2i-1}$, $r'(\bar{x}_i) = t_{2i}$.

Now if the instance of 3-SAT is satisfiable, we can solve the instance of VZ as follows:

For $1 \leq i \leq n$, if x_i is **true**,
let $P(t_{2i-1}) = 4i-3$, $P(f_{2i-1}) = 4i-2$, $P(f_{2i}) = 4i-1$, and $P(t_{2i}) = 4i$;
if x_i is **false**,
let $P(f_{2i-1}) = 4i-3$, $P(t_{2i-1}) = 4i-2$, $P(t_{2i}) = 4i-1$, and $P(f_{2i}) = 4i$.
For $1 \leq i \leq m$, let $P(t_{2n+3i}) = 4n+6i$,
and define $P(x) = 4n + \Delta(x) + 6i - 6$ on the other items according to the table in figure 3.

It is straightforward to verify that P satisfies all the conditions required for an affirmative answer to the instance of VZ.

Conversely, suppose we have solved the instance of VZ in the affirmative. We will construct the solution to the 3-SAT instance that shows that it is satisfiable. The first thing to do is to verify that the VZ solution must fall into a pattern as outlined above. Every page must contain exactly one item, because the page size is to small to allow two or more items on a page, and the pages are not allowed to be empty. Furthermore, the ordering and page size constraints on P force the layout to satisfy

$\{P(t_{2i-1}), P(f_{2i-1})\} = \{4i-3, 4i-2\}$ and
$\{P(t_{2i}), P(f_{2i})\} = \{4i-1, 4i\}$ for $1 \leq i \leq n$, and
$\{P(f_{2n+3i-2}), P(f_{2n+3i-1}), P(f_{2n+3i}), P(t_{2n+3i-2}), P(t_{2n+3i-1})\} =$
$\{4n+6i-5, \ldots, 4n+6i-1\}$
and $P(t_{2n+3i}) = 4n+6i$ for $1 \leq i \leq m$.

u_i	v_i	w_i	$\Delta(f_{2n+3i-2})$	$\Delta(f_{2n+3i-1})$	$\Delta(f_{2n+3i})$	$\Delta(t_{2n+3i-2})$	$\Delta(t_{2n+3i-1})$
true	true	true	1	3	5	2	4
true	true	false	1	3	4	2	5
true	false	true	1	2	3	4	5
true	false	false	1	2	4	3	5
false	true	true	2	3	5	1	4
false	true	false	2	3	4	1	5
false	false	true	2	4	5	1	3

Figure 3. Embedding function for clauses of VZ.

Because of the references from t_{2i-1} to f_{2i} and from t_{2i} to f_{2i-1}, the pairs (f_{2i-1}, t_{2i-1}) and (t_{2i}, f_{2i}) must be placed so that the figure comes first in exactly one of them. For $1 \leq i \leq m$, at least one of $f_{2n+3i-2}$, $f_{2n+3i-1}$, or f_{2n+3i} must fall on an odd page, meaning its corresponding literal must be true. Each of these figures has a reference from a text block in the first $4n$ pages that forces its truth value (its page number modulo 2) to be the same as all other figures that represent the same literal. Thus if we let $x_i =$ true if $P(t_{2i})$ is odd and false otherwise, for $1 \leq i \leq n$, then at least one literal in each clause will be true, and the formula is satisfiable. This completes the proof of theorem 3.

CONSTANT PAGE SIZE

We can also prove that VZ is NP-complete even if the page size is constant. The construction is similar to the one above, but a different method is used to constrain the possible positions of the figures. Each figure will be smaller than the page size, but greater than half the page size so that two figures cannot occur on the same page. The balance of each page with a figure on it must be filled up with one or more text blocks; by supplying the proper sequence of text block sizes, we will be able to force the figures to be placed on only the pages we wish.

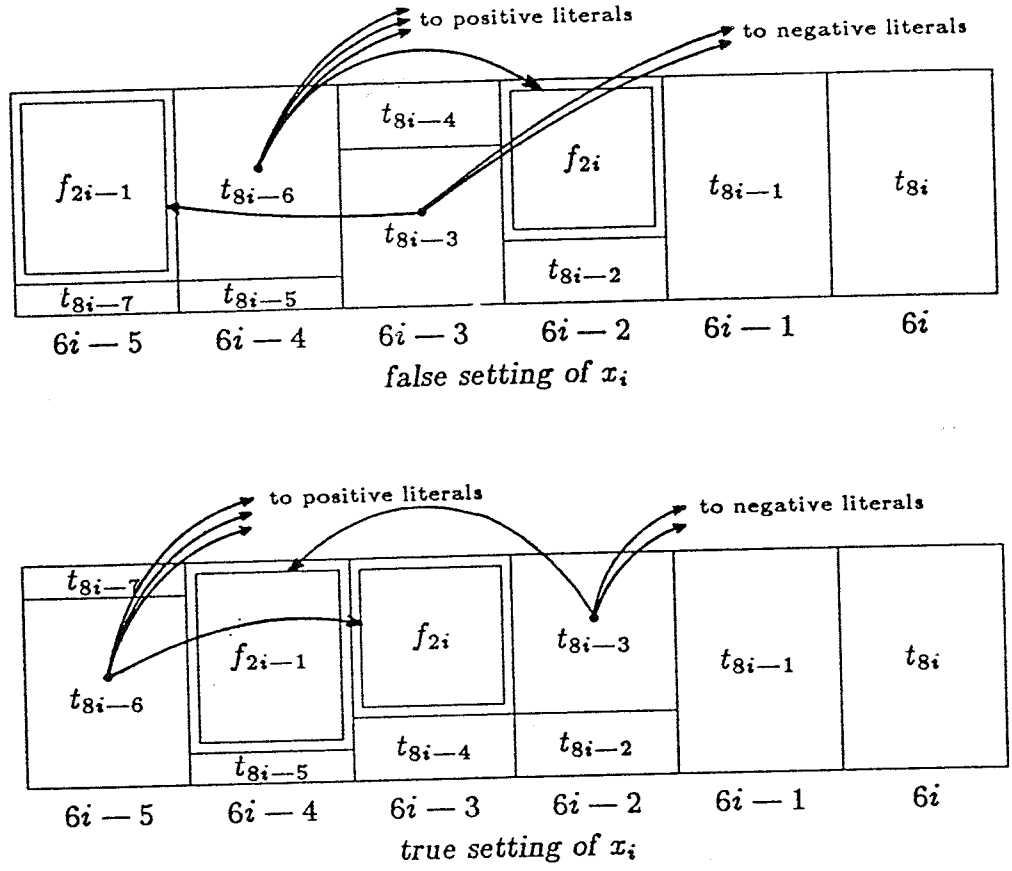


Figure 4. Truth value settings for VZ *with constant page size.*

Theorem 4. VZ is NP complete, even if the page size $s(i)$ is constant.

Proof: We will reduce an instance of 3-SAT to an instance of VZ with the constant page size function $s(i) = 7$.

For each variable x_i, $1 \leq i \leq n$, we will introduce 8 text blocks and 2 figures. The heights h of the text blocks $t_{8i-7}, t_{8i-6}, \ldots, t_{8i}$ are 1,6,1,2,5,2,7, and 7, respectively, and the heights of the figures are $h(f_{2i-1}) = 6$ and $h(f_{2i}) = 5$. As before, there are references tying each of these figures to a text block: $r(f_{2i-1}) = t_{8i-3}$ and $r(f_{2i}) = t_{8i-6}$. The intention is that these text blocks and figures be placed on pages $6i-5, 6i-4, \ldots, 6i$ in one of the two ways shown in figure 4.

We need to show that these are the only two alternatives that contribute no badness. We may assume by induction that the previous figures and text blocks were placed in the previous $6(i-1)$ pages. The text blocks t_{8i-1} and t_{8i} must fall on consecutive pages, and the other text blocks $t_{8i-7}, \ldots, t_{8i-2}$, along with whatever figures that fall before t_{8i-1} must have a total height that is an integral multiple of the page size. Thus one or both of f_{2i-1} and f_{2i} cannot come after t_{8i-1}, or else this total height would be 23 or 17, neither of which is a multiple of 7. Furthermore, no later figures may be placed before t_{8i}, because this would require that f_{2i+1}, which has height 6, be placed on a page with either t_{8i-7} or t_{8i-5}; this is not possible since t_{8i-6} and f_{2i-1} must each share a page with one of these. Now to attain zero badness, t_{8i-6} and f_{2i} must fall on different pages with the same handedness, and so the only two ways to place these items are those shown above.

Next we show how to encode the clauses. Let $r'(x_i) = t_{8i-6}$ and $r'(\bar{x}_i) = t_{8i-3}$, for $1 \leq i \leq n$. The clause u_j, v_j, w_j, for $1 \leq j \leq m$, will be encoded by 12 text blocks $t_{8n+12j-11}, \ldots, t_{8n+12j}$ of heights 1,1,1,4,1,1,1,4,1,1,1,7, respectively, and by 3 figures $f_{2n+3j-2}, f_{2n+3j-1}, f_{2n+3j}$, all of size 6, with $r(f_{2n+3j-2}) = r'(u_j)$, $r(f_{2n+3j-1}) = r'(v_j)$, and $r(f_{2n+3j}) = r'(w_j)$.

These 12 text blocks and 3 figures must appear on 6 consecutive pages, by an argument similar to what we have just considered, because if any of the 3 figures occur after the last text block, the sum of the heights before the last text block is not divisible by 7. No figures may be introduced from the encoding of the following clause, because three of the small text blocks must occur on the same pages as the three figures, and the remaining 6 small text blocks are needed to pad the pages that contain the text blocks of height 4; thus no small text blocks are left to pair up with the extra figures.

Now given a solution to the instance of VZ, the solution to the instance of 3-SAT may be constructed as before, by setting $x_i =$ **true** if $P(t_{8i-6})$ is odd, and **false** otherwise, for $1 \leq i \leq n$. Conversely, if we have a solution to the 3-SAT instance, we can construct a solution to the VZ instance as follows:

The items $\{t_1, \ldots, t_{8n}\}$ and $\{f_1, \ldots, f_{2n}\}$ are mapped as in figure 4, according to the setting of the x_i. To explain the mapping of the remaining items, we will introduce a new notation. Square brackets will enclose two rows of numbers, the top row containing the sizes of the figures and the bottom row containing the sizes of the text blocks. Vertical bars will mark the boundaries between the pages. Since the order of the text and the order of the figures are both fixed, this notation unambiguously specifies an

u_j	v_j	w_j	*embedding*
true	true	true	$\left[\begin{array}{c\|cccc\|c\|cccc\|c\|c} 6 & & & & & 6 & & & & & 6 & \\ 1 & 1 & 1 & 4 & 1 & 1 & 1 & 4 & 1 & 1 & 1 & 7 \end{array}\right]$
true	true	false	$\left[\begin{array}{c\|cccc\|c\|c\|cccc\|c} 6 & & & & & 6 & 6 & & & & \\ 1 & 1 & 1 & 4 & 1 & 1 & 1 & 4 & 1 & 1 & 1 & 7 \end{array}\right]$
true	false	true	$\left[\begin{array}{c\|c\|c\|cccc\|cccc\|c} 6 & 6 & 6 & & & & & & & & & \\ 1 & 1 & 1 & 4 & 1 & 1 & 1 & 4 & 1 & 1 & 1 & 7 \end{array}\right]$
true	false	false	$\left[\begin{array}{c\|c\|cccc\|c\|cccc\|c} 6 & 6 & & & & & 6 & & & & & \\ 1 & 1 & 1 & 4 & 1 & 1 & 1 & 4 & 1 & 1 & 1 & 7 \end{array}\right]$
false	true	true	$\left[\begin{array}{cccc\|c\|c\|cccc\|c\|c} & & & & 6 & 6 & & & & & 6 & \\ 1 & 1 & 1 & 4 & 1 & 1 & 1 & 4 & 1 & 1 & 1 & 7 \end{array}\right]$
false	true	false	$\left[\begin{array}{cccc\|c\|c\|c\|cccc\|c} & & & & 6 & 6 & 6 & & & & & \\ 1 & 1 & 1 & 4 & 1 & 1 & 1 & 4 & 1 & 1 & 1 & 7 \end{array}\right]$
false	false	true	$\left[\begin{array}{cccc\|c\|cccc\|c\|c\|c} & & & & 6 & & & & 6 & 6 & & \\ 1 & 1 & 1 & 4 & 1 & 1 & 1 & 4 & 1 & 1 & 1 & 7 \end{array}\right]$

Figure 5. Embeddings for the various possible truth values.

embedding. Using this notation, the mapping of the remaining items is as shown in figure 5.

If the mapping P is chosen in this way, it will satisfy all the properties specified in the definition of VZ; this completes the proof of theorem 4.

ORDERED FIGURES

In the previous three constructions there was exactly one citation for each figure, but the order specified for the figures was not the same as the order of the citations. It seems natural to speculate that if we required them to be in the same order, the problem would be easier to solve. The next theorem will show that this is not the case, at least when we allow glue between the text blocks.

The model we shall use incorporates most of the features of the boxes and glue model. We will not need penalty nodes, and the glue nodes will have no stretch or shrink components.

Page Breaking Problem with Ordered Figures and with Zero-one Badness Function (OZ):

Given: A sequence $t_1, t_2, \ldots, t_N$ of nonnegative integers, the text block sizes;
a sequence $g_1, g_2, \ldots, g_{N-1}$ of nonnegative integers, the glue sizes;
a sequence $f_1, f_2, \ldots, f_M$ of nonnegative integers, the figure sizes;
a nondecreasing function $r : \{1, 2, \ldots, M\} \to \{1, 2, \ldots, N\}$, telling what text block cites each figure;
and a positive integer c, the page size.

Question: Do there exist an integer p and nondecreasing functions
$P : \{1, 2, \ldots, N\} \to \{1, 2, \ldots, p\}$ and
$Q : \{1, 2, \ldots, M\} \to \{1, 2, \ldots, p\}$ such that, for $1 \leq k \leq p$,

$$\sum_{i_0 \leq i \leq i_1} t_i + \sum_{i_0 \leq i < i_1} g_i + \sum_{j_0 \leq j \leq j_1} f_j = c,$$

where $i_0 = \min\{\, i \mid P(i) = k\}$, $i_1 = \max\{\, i \mid P(i) = k\}$,
$j_0 = \min\{\, j \mid Q(j) = k\}$, $j_1 = \max\{\, j \mid Q(j) = k\}$;
and $P(r(j)) \equiv Q(j) \pmod 2$ for $1 \leq j \leq M$?

Theorem 5. OZ is NP-complete.

Proof: The reduction is again from 3-SAT. The construction is more complicated than for the earlier proofs, so we will view it at a higher level of abstraction to understand more easily how it works.

The construction will be built up out of modules, each consisting of a sequence of figure sizes and a sequence of alternating text block and glue sizes. These modules will be joined together to form the text, glue, and figure sequences that define the instance of OZ. A module containing figures of sizes $f_1, f_2, \ldots, f_i$, text blocks of sizes $t_1, t_2, \ldots, t_j$, and glue items of sizes $g_1, g_2, \ldots, g_{j-1}$ will be written as

$$\left\{ \begin{array}{c} f_1\, f_2 \cdots f_i \\ t_1[g_1]t_2[g_2]\cdots[g_{j-1}]t_j \end{array} \right\}.$$

The intention is that the contents of a module will fall into a range of pages in the embedding that will include no other text or figures. To ensure this, we will choose the figure sizes so that they are all different and occupy almost

all of a page. The rest of each page that contains a figure will be filled up by a *shim*, that is, a text block whose size adds to the figure's size to make up the page size. The shim sizes will be carefully chosen to prevent two or more of them from combining with a figure in the place of a shim that was intended for the figure. This approach is quite conservative, since in reality we only have to worry about figures straying into neighboring modules, but we will stay with it since it simplifies the proof.

To make sure that each module begins and ends on a page boundary, we will constrain the first and last text block sizes in each module to be equal to the page size c.

It is convenient to use a comma to stand for zero glue [0], since it comes up so often in the construction. In addition, we will use $\bar{\alpha}$ to stand for the expression $c - \alpha$.

When joining modules together, we need to specify what text blocks reference the figures in the module. We will use the convention that only one text block in each module may reference any figure, and this text block will be distinguished by underlining it in the description of the module. Writing

$$M_i = M^\tau(k_1, k_2, \ldots)$$

means that the module i is of type τ and the first figure in M_i is referenced by the distinguished text block in module k_1, the second figure by the distinguished text block in module k_2, and so on. When a module is appended onto a sequence of modules, the figure sequence of the new module is appended onto the previous figure sequence, the text sequence of the module is appended onto the previous text sequence, and the glue sequence is extended on the front by zero glue and then appended onto the previous glue sequence.

We will denote 'odd' by **1** and 'even' by **0**. The construction will represent true by **1** and false by **0**.

Each module has one *horizontal input*, one *horizontal output*, one *vertical input* for each figure it contains, and one *vertical output*. Each of these inputs and outputs may be **1** or **0**, and for most of the modules the outputs are a function of the inputs. The horizontal input is **0** or **1** depending on whether the module begins on an even or odd page in the embedding, and the horizontal output is **0** or **1** depending on whether the next module after it in the embedding begins on an even or odd page. The values of the vertical inputs depend on the placements of the text blocks that reference the figures

in the module and the value of the vertical output depends on the placement of the distinguished text block.

Let us take a look at some module types that we will need later. An *inverter* looks like this:

$$M^{\sim} = \left\{ \begin{matrix} \bar{\alpha} \\ c, \alpha, \underline{\bar{\alpha}}, \alpha, c \end{matrix} \right\}.$$

The lowercase greek letters signify a shim size, and their use is local to a module definition. Shim sizes defined in different modules will be different, even if the same letter is used to represent them in both module definitions. The module $M^{\sim}$ will pass the value of the horizontal input to the horizontal output unchanged, and the value of the vertical output will be the complement of the vertical input. This may be verified by examining the two possible embeddings of $M^{\sim}$:

$$\left[\begin{matrix} c \end{matrix} \middle| \begin{matrix} \bar{\alpha} \\ \alpha \end{matrix} \middle| \begin{matrix} \underline{\bar{\alpha}}, \alpha \end{matrix} \middle| \begin{matrix} c \end{matrix} \right] \quad \text{or} \quad \left[\begin{matrix} c \end{matrix} \middle| \begin{matrix} \alpha, \underline{\bar{\alpha}} \end{matrix} \middle| \begin{matrix} \bar{\alpha} \\ \alpha \end{matrix} \middle| \begin{matrix} c \end{matrix} \right].$$

One of these embeddings is forced by each of the four possible combinations of the inputs, and in each case the output is correct.

The *buffer* module passes the variables unchanged in both directions:

$$M^{e} = \left\{ \begin{matrix} \bar{\alpha} \\ c, 0[\bar{1}]1, \underline{\alpha - 1}, 1[\bar{1}]0, c \end{matrix} \right\},$$

where α and $\alpha - 1$ are shim sizes not used elsewhere.

The two possible embeddings for M^{e} are

$$\left[\begin{matrix} c \end{matrix} \middle| \begin{matrix} 0[\bar{1}]1 \end{matrix} \middle| \begin{matrix} \bar{\alpha} \\ \underline{\alpha - 1}, 1 \end{matrix} \middle| \begin{matrix} 0, c \end{matrix} \right] \quad \text{or} \quad \left[\begin{matrix} c, 0 \end{matrix} \middle| \begin{matrix} \bar{\alpha} \\ 1, \underline{\alpha - 1} \end{matrix} \middle| \begin{matrix} 1[\bar{1}]0 \end{matrix} \middle| \begin{matrix} c \end{matrix} \right],$$

and both of these fill an even number of pages and have the distinguished text block on the same page as the figure.

The *generator* passes the horizontal input unchanged, has no vertical input, and can have a vertical output of either 0 or 1. The generator is useful for making a nondeterministic choice for the variables when solving

the 3-SAT problem:

$$M^? = \left\{ \begin{matrix} \\ c, \underline{0}, c \end{matrix} \right\}.$$

The *junction* module has no vertical input, and passes the value of its horizontal input to both of its outputs:

$$M^\top = \left\{ \begin{matrix} \\ \underline{c}, c \end{matrix} \right\}.$$

When the vertical output is not used, this becomes the null module,

$$M^\emptyset = \left\{ \begin{matrix} \\ c, c \end{matrix} \right\},$$

useful as a place holder when specifying how to put modules together.

The *test* module may be embedded only with the horizontal and vertical inputs the same; the horizontal output is the complement of the input, and the vertical output is the same as the input:

$$M^= = \left\{ \begin{matrix} \bar{\alpha} \\ c, c, \underline{\alpha}, c, c \end{matrix} \right\}.$$

The only other module we need is the *or-gate* $M^\vee$. The description of $M^\vee$ is rather complicated, and will be deferred until after we have seen how the whole construction fits together. The $M^\vee$ module has two vertical inputs; the first of these will always be 1 in our construction. The vertical output of $M^\vee$ is not used, and the horizontal output is given by this table:

horizontal input	*second vertical input*	*horizontal output*
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**1**

The first vertical input is needed as a reference, for without it the module has no way of knowing what is even and what is odd.

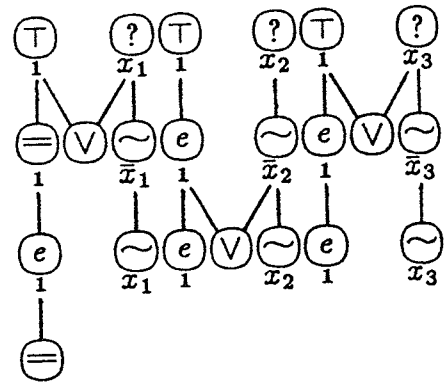


Figure 6. Embedding for a small example of OZ.

Now we can put the modules together to form an instance of OZ that will have the same answer as the given instance of 3-SAT. The modules will be arranged in $2m+2$ rows, each sending its output variables to the row below. The easiest way to see how the construction works is to look at an example.

Suppose $X = \{x_1, x_2, x_3\}$ and $C = \{x_1 \vee \bar{x}_2 \vee x_3\}$. Then the construction may be diagramed as in figure 6.

The first row guesses the settings of the variables, and the remaining rows check to see if these settings will satisfy the clauses. The encoding of each clause occupies two rows. The first of these starts with an $M^{=}$ module, which checks that the previous clause was true, and forces the module that follows it to begin on an even page; in other words, it forces the horizontal variable to **0**. The only other modules in the two rows are the three or-gates. These will set the horizontal output to **1** if the horizontal input was **1** or if the vertical input, which is attached to a module corresponding to one of the literals, is **1**. Thus at the end of row $2i+1$, the horizontal output corresponds to the truth value of clause i. These outputs must all be 1 in a valid embedding, since they are each followed by a test module to make sure this happens.

More precisely, the construction goes like this: We will think of the sequence of modules $\{M_1, M_2, \ldots, M_{6mn+3n+1}\}$ as an array with $2m+2$ rows, n columns, and 3 modules for each entry. To aid in this identification, we define $\langle i, j, k\rangle = 6ni + 3j + k + 1$ for $0 \le i \le 2m+1$, $0 \le j < n$, and $0 \le k < 3$.

Now we define

$$M_{\langle 0,j,0\rangle} = M^{\top},\ M_{\langle 0,j,2\rangle} = M^{?} \quad \text{for} \quad 0 \le j < n,$$
$$M_{\langle 2i+1,0,0\rangle} = M^{=}(\langle 2i,0,0\rangle) \quad \text{for} \quad 0 \le i \le m,$$
$$M_{\langle 2i+1,j,0\rangle} = M^{e}(\langle 2i,j,0\rangle) \quad \text{for} \quad 0 \le i < m,\ 1 \le j < n,$$
$$M_{\langle 2i,j,0\rangle} = M^{e}(\langle 2i-1,j,0\rangle) \quad \text{for} \quad 1 \le i \le m,\ 0 \le j < n,$$
$$M_{\langle i,j,2\rangle} = M^{\sim}(\langle i-1,j,2\rangle) \quad \text{for} \quad 1 \le i \le 2m,\ 0 \le j < n.$$

If x_j appears in clause i, let

$$M_{\langle 2i-1,j-1,1\rangle} = M^{\vee}(\langle 2i-2,j-1,0\rangle,\langle 2i-2,j-1,2\rangle).$$

If $\bar{x}_j$ appears in clause i, let

$$M_{\langle 2i,j-1,1\rangle} = M^{\vee}(\langle 2i-1,j-1,0\rangle,\langle 2i-1,j-1,2\rangle).$$

Let all the M_i not otherwise defined be $M^{\emptyset}$ for $1 \le i \le 6mn+3n+1$.

The figure, text, and glue sequences for the instance of OZ are obtained by concatenating these modules together in order, with zero glue being inserted whenever two lower lists are combined, as described above. The reference function r is created as indicated by the module definitions.

These are still two loose ends in the construction that have to be explained. One is how to pick the shim sizes, and the other is how to build an or-gate module. Some care is needed in picking the shim sizes. Suppose we need l different shim sizes; $l = O(mn)$ since there are $O(mn)$ modules and the number of different shims that any module needs is bounded by a constant. Then we can pick the shim sizes to be $l+1, l+2, \ldots, 2l$. Any two or more shims put together will then be larger than any single shim, and will not be able to take the place of a single shim in the construction. The page size c may then be picked to be at least $4l+1$, so that each figure occupies at least half of a page, implying no two figures can appear on the same page.

Finally, we will look at how to implement an or-gate module. The top line of this module will contain the two figure sizes $\bar{\alpha}$ and $\bar{\beta}$. The bottom line of the module will consist of four parts, each part handling one of the four possible input configurations (recalling that the figure of size $\bar{\alpha}$ will always fall on an odd page). Both figures must fall into the same one of these four parts when the module is embedded, by the way the parts are constructed. The part that gets the two figures will occupy an odd or even number of

pages, whichever is needed to yield the correct horizontal output, and the other three parts will each occupy an even number of pages, thus leaving the horizontal variable unchanged.

The first part accepts the figures exactly when it starts on an even page and the second figure also falls on an even page. Its text and glue sequence and its two possible embeddings are

$$\sigma_{00} = c,\alpha\overline{[\alpha+\beta+\gamma]}\beta,\gamma[\bar{\gamma}]0,c,c \qquad \left\{ \begin{array}{l} \left[c \left| \begin{matrix} \bar{\alpha} \\ \alpha \end{matrix} \right| \begin{matrix} \bar{\beta} \\ \beta \end{matrix} \right| \gamma[\bar{\gamma}]0 \left| c \right| c \right] \\ \left[c \left| \alpha\overline{[\alpha+\beta+\gamma]}\beta,\gamma \right| 0,c \left| c \right| \right] \end{array} \right.$$

The remaining three parts work analogously:

$$\sigma_{01} = c,\alpha\overline{[\alpha+\gamma]}\gamma\overline{[2\gamma]}\gamma\overline{[\gamma+\beta]}\beta,c \qquad \left\{ \begin{array}{l} \left[c \left| \begin{matrix} \bar{\alpha} \\ \alpha \end{matrix} \right| \gamma\overline{[2\gamma]}\gamma \left| \begin{matrix} \bar{\beta} \\ \beta \end{matrix} \right| c \right] \\ \left[c \left| \alpha\overline{[\alpha+\gamma]}\gamma \right| \gamma\overline{[\gamma+\beta]}\beta \right| c \right] \end{array} \right.$$

$$\sigma_{10} = c,c,\alpha\overline{[\alpha+\beta+\gamma]}\beta,\gamma[\bar{\gamma}]0,c \qquad \left\{ \begin{array}{l} \left[c \left| c \left| \begin{matrix} \bar{\alpha} \\ \alpha \end{matrix} \right| \begin{matrix} \bar{\beta} \\ \beta \end{matrix} \right| \gamma[\bar{\gamma}]0 \left| c \right| \right] \\ \left[c \left| c \left| \alpha\overline{[\alpha+\beta+\gamma]}\beta,\gamma \right| 0,c \right| \right] \end{array} \right.$$

$$\sigma_{11} = c,c,\alpha\overline{[\alpha+\gamma]}\gamma\overline{[2\gamma]}\gamma\overline{[\beta+2\gamma]}\beta,\gamma[\bar{\gamma}]0,c,c \qquad \left\{ \begin{array}{l} \left[c \left| c \left| \begin{matrix} \bar{\alpha} \\ \alpha \end{matrix} \right| \gamma\overline{[2\gamma]}\gamma \left| \begin{matrix} \bar{\beta} \\ \beta \end{matrix} \right| \gamma[\bar{\gamma}]0 \left| c \right| c \right] \\ \left[c \left| c \left| \alpha\overline{[\alpha+\gamma]}\gamma \right| \gamma\overline{[\beta+2\gamma]}\beta,\gamma \right| 0,c \left| c \right| \right] \end{array} \right.$$

The or-gate module is then

$$M^{\vee} = \left\{ \begin{matrix} \bar{\alpha} & \bar{\beta} \\ \sigma_{00}, \sigma_{01}, \sigma_{10}, \sigma_{11} \end{matrix} \right\}.$$

This completes the construction for the proof that OZ is NP-complete.

SUMMARY To conclude this chapter, we will step back from the morass of details in the proofs herein, and try to give an intuitive explanation of why these particular problems are NP-complete. The single most important factor seems to be the badness function; both of the functions used here allow information to be communicated over long distances in the embedding, without affecting the portion of the embedding in between. In addition, the constructions rely on various mechanisms for restricting the possible positions of each figure to a small range of pages. Some of the constructions used multiply-referenced figures for this requirement; others exploited the ability to have more than one item per page, a variable page size, the presence of glue, or a combination of these. Something of this nature is needed to make the construction work; for example a problem with a quadratic badness function, one item per page, one reference per figure, and the figures in the same order as the citations would be easily solved by putting each figure right after its citation. If we instead allow multiple items per page with a constant page size and with glue, the NP-completeness of the resulting problem is an open question. Other open problems could be generated by taking the various elements of the problems in this chapter in different combinations, but the proofs we have already seen give a sufficient indication that there are severe restrictions on the kind of badness function that can be used in a practical pagination scheme.

Chapter 3

Dynamic Programming for Pagination

The first requisite in all book design is orderliness.
— B. Rogers

ONE OF THE MOST SUCCESSFUL WAYS we have of solving a large problem is to break it into smaller problems, solve each one of those individually, and then piece the solutions together to find the solution to the original problem. This approach may be applied recursively to the smaller problems, breaking them down further and further until at the lowest level we need to solve only trivial problems, albeit we may have to solve a great many of them. For many problems this recursive decomposition may be applied directly with good results. However, it often happens that the same subproblems arise many times when breaking down problems in this way, and we do the work of solving them again and again. The way to avoid this extra work is to remember the solution to each subproblem as it is solved, so that when it is needed again later on, it won't have to be recomputed. If we know what all the subproblems will look like in advance, the easiest way to manage the bookkeeping is to work from the bottom up, solving the trivial problems first, then those that decompose into only trivial problems, and so forth until we get the answer to the problem we originally wanted to solve. This technique is called *dynamic programming*, a term coined by R. Bellman [14], who first popularized it as a general technique for solving optimization problems.

As the first example, we shall see how to apply dynamic programming to the pagination problem when there are no figures involved. In this case the problem is completely analogous to the problem of breaking a paragraph into

lines, a problem considered in detail by Knuth and Plass [9]. Here we will retain the terminology of the pagination problem, and stay with a high-level description of the solution, leaving the details for the next chapter.

PAGINATION WITH NO FIGURES

The idea is to consider subproblems of the form "find the best way to put the first j lines, along with the interspersed glue, onto the first k pages." To this end, we will define B_{jk} to be the total badness corresponding to the best way to put lines 1 thru j on pages 1 thru k. Now suppose we have such an optimal pagination with total badness B_{jk}, and let i be the number of the last line on the next to last page, so that page k contains lines $i+1$ thru j. Then the total badness for pages 1 thru $k-1$ must be exactly $B_{i,k-1}$, because if it were less, $B_{i,k-1}$ would not be the smallest badness possible, and if it were greater, B_{jk} would not be the minimum. The important point here is that the way the material is arranged in the first part of the document does not interact with the placement of the rest of the material; the problem is said to satisfy the *principle of optimality*, and this is what makes dynamic programming applicable to this problem.

There are not too many ways to pick i, so one way to compute B_{jk} is to use the formula

$$B_{jk} = \min_{0 \le i < j} \{B_{i,k-1} + \beta_{ijk}\}$$

where β_{ijk} is the badness due to placing lines $i+1$ thru j on page k. The quantities β_{ijk} may each be computed in constant time, if the subtotals for the box sizes and glue have been precalculated.

It is not hard to see why this formula works, but there are still a lot of details to consider before this can be made into a practical solution to the pagination problem. One of these is the question of how to extract the actual optimum pagination, instead of just its badness. To do this it is only necessary to store, along with the value of B_{jk}, the value of i that minimized the sum $B_{i,k-1} + \beta_{ijk}$. Once the winning solution for the entire problem has been chosen, these may be used as links in a chain to trace back and find out what lines to put on each page.

IMPROVING THE RUNNING TIME

If this method is applied directly by calculating all of the B_{jk}, the running time will be $\Theta(n^2)$ if there are n lines in the input. The constant factor on this running time is fairly high, so in its simplest form the algorithm is too slow to be routinely applied to practical pagination problems. However, most of the numbers B_{jk} will be so large that we are not interested in them. If this fact is exploited to limit the number of subproblems that are considered,

the running time can be decreased to the point where it can be considered practical. The details of this pruning technique will be saved until chapter four, since we are concerned primarily with theoretical issues in the present chapter.

There is an important special case for which the running time may be reduced even more dramatically. If the page size is constant and the same badness function applies for each page, so that the β_{ijk} do not depend on the page number k, then we only have to remember the best way to put the first j lines into any number of pages. This happens because if we have a way to put, say, the first 1000 lines of text onto 20 pages and a way to put them onto 21 pages, in the optimal solution we will use whichever one of them has a smaller badness, because this choice cannot affect the way we choose to paginate the rest of the document. Since we will only use the better of these partial solutions, we may as well remember only one of them.

More formally, if B_j is the lowest total badness for paginating the first j lines, and if β_{ij} is the badness of putting lines $i+1$ thru j onto a page, then

$$B_j = \min_{0 \le i < j} \{B_i + \beta_{ij}\}.$$

If there are n lines, this calculation takes $O(n)$ space and $O(n^2)$ time; normally the height of each line is bounded below, so that each page can hold at most c lines, for some constant c. In this case, the minimum need be taken only over the range $j - c \le i < j$, so the running time will be proportional to cn.

INCLUDING FIGURES

When figures are introduced, care must be taken when defining the badness function for figure placement so that the resulting problem is amenable to a solution by dynamic programming. Chapter two presented some badness functions to be avoided; the property those functions share is that they can be used to communicate information over long distances in the embedding, and this is also the property we need to avoid in order to satisfy the principle of optimality. We have seen that the proof that MQ is NP-complete does not work if the quadratic badness function is replaced by a linear badness function, i.e., if the square in the badness function is replaced by the absolute value. This suggests that perhaps the problem with the linear badness function may have a polynomial time solution, and it turns out that it does.

To be precise, let us define the

Page Breaking Problem with Multiply Referenced Figures and Linear Badness Function (ML):

GIVEN: A set $T = \{t_1, t_2, \ldots, t_N\}$ of text blocks,
a set $F = \{f_1, f_2, \ldots, f_M\}$ of figures, with $T \cap F = \emptyset$,
a mapping $W : T \times F \to \{0, 1, 2, \ldots\}$, and a bound B.

QUESTION: Is there a 1-1 mapping $P : (T \cup F) \to \{1, 2, \ldots, N + M\}$
such that $P(t_i) < P(t_j)$, $P(f_i) < P(f_j)$ for $i < j$, and

$$\sum_{\substack{1 \le i \le N \\ i \le j \le M}} W(t_i, f_j)\,|P(t_i) - P(f_j)| \le B?$$

This problem can be solved by decomposing it into problems of the form "what is the best way to put the first i text blocks and the first j figures into the first $i + j$ pages?" In order for this question to make sense, we have to extend the badness function so that it can be applied to just part of the input. Suppose we have merged the first i text blocks and the first j figures into the first k pages in some way. Some of the text blocks may reference figures that are not in this initial segment, and similarly some of the first j figures might be referenced by text blocks that occur later. These "dangling references" must be accounted for in some way when computing the badness of the partial embedding. One way to think of the calculation of the badness is to think of a penalty being charged every time a reference crosses over a page boundary. (Recall that a reference can be thought of as an arc leading from a citation to its figure.) Since the dangling references are known to end somewhere past the end of the partial embedding, it makes sense to include only the page boundaries they have already crossed when calculating the badness of the partial embedding. Using this idea, we can solve the ML problem as follows:

Let $R_{i,j}$ be the number of dangling references when the first i text blocks and the first j figures have been placed on the first $i+j$ pages. This function is well-defined, because any rearrangement of the figures and text on the first k pages does not affect the set of references that must cross the boundary between page k and page $k+1$. The number of dangling forward references is given by the sum

$$\sum_{\substack{1 \le r \le i \\ j < s \le M}} W(t_r, f_s),$$

since we want to count the edges that originate in the first part of the embedding and end in the last part. Similarly, the number of dangling backward references is given by the sum

$$\sum_{\substack{i<r\leq N \\ 1\leq s\leq j}} W(t_r, f_s).$$

The value of $R_{i,j}$ could be calculated by just adding these two sums together, but we can save some calculation by first tabulating the sums

$$S_{ij} = \sum_{\substack{1\leq r\leq i \\ 1\leq s\leq j}} W(t_r, f_s)$$

and then calculating the dangling reference count by the formula

$$R_{ij} = S_{iM} + S_{Nj} - 2S_{ij}.$$

It is not hard to verify this formula by considering the regions over which the summations occur.

The badness B_{ij} may now be calculated according to the formula

$$B_{ij} = \min(B_{i-1,j}, B_{i,j-1}) + R_{ij}$$

with the boundary conditions $B_{00} = 0$ and $B_{ij} = \infty$ if either i or j is negative. Choosing $b_{i-1,j}$ as the smaller of the two numbers corresponds to putting text block i on page $i+j$, and choosing $B_{i,j-1}$ corresponds to putting figure j on page $i+j$. The term R_{ij} accounts for the penalty due to the dangling edges crossing the boundary between pages $i+j$ and $i+j+1$. Then B_{NM} is the smallest possible total badness for embedding all N text blocks and all M figures.

To get an embedding that achieves this minimum badness, we need only trace a path through the table of badness values that starts at B_{NM}, ends at B_{00}, and at each step goes from some B_{ij} to the smaller of $B_{i-1,j}$ and $B_{i,j-1}$. If the path goes from B_{ij} to $B_{i-1,j}$, then $P(t_i) = i+j$, and if it goes from B_{ij} to $B_{i,j-1}$, then $P(f_j) = i+j$. Note that there might be many embeddings that all achieve the minimum badness.

MAKING THE ALGORITHM MORE GENERAL

This algorithm may be generalized to allow references that join text blocks to each other, rather than to figures, and to allow a third sequence of items that is to be merged in as well; this could be, for example, a sequence of tables that are referenced in the text just like figures are, but that are numbered independently of the figures. To handle three sequences, it is necessary to solve a three-dimensional array of subproblems, and the exponent on the running time increases accordingly.

A further generalization can be made if we introduce the notion of partial ordering. A *partially ordered set* $(V, \prec)$ is a set V along with a binary relation $\prec$ that satisfies the following properties, for all $u, v, w \in V$:

1. $u \not\prec u$ *(irreflexivity)*
2. $u \prec v$ and $v \prec w$ imply that $u \prec w$ *(transitivity).*

Note that this is like a total ordering, except that it is not required that every two elements be related to each other. If $u \prec v$ or $u = v$, we write $u \preceq v$. If $u \not\prec v$, $v \not\prec u$, and $u \neq v$, then we say the elements u and v are *incomparable.* If $u_1 \prec u_2 \prec \ldots \prec u_k$, then the sequence $\{v_1, v_2, \ldots, v_k\}$ is called a *chain*; if the elements of a set $U \subseteq V$ are pairwise incomparable, then U is an *antichain.*

Using this notion of partial ordering, we can define the following problem:

K-Constrained Optimal Linear Arrangement (K-COLA):

GIVEN: A partially ordered set $(V, \prec)$ of vertices, such that the size of the largest antichain does not exceed K,
a graph $G = (V, E)$,
an edge weight function $W : E \to \{0, 1, 2, \ldots\}$,
and a bound C.

QUESTION: Is there a one-to-one function $P : V \to \{1, 2, \ldots, |V|\}$ such that $P(u) < P(v)$ whenever $u \prec v$,

$$\text{and} \sum_{(u,v)\in E} w((u,v))\,|f(u) - f(v)| \leq C?$$

If the size of the antichains is not constrained, the problem is NP-complete, as shown by Garey, Johnson, and Stockmeyer [13]. Even and Shiloach showed that this is the case even if G is a bipartite graph [15].

The ML problem is easily seen to be an instance of K-COLA in which the partial order consists of two chains, one containing the figures and the other

containing the text blocks. In this case the graph G is bipartite, with the figures on one side and the text blocks on the other.

We will call a subset X of vertices a *leading subset* if $x \in V$ and $y \in X$ and $x \prec y$ together imply that $x \in X$. (Another name for a leading subset is an *order ideal*.) Every antichain of V will have a corresponding leading subset of V, obtained by starting with the antichain and successively adding elements that precede elements already in the set until no more such elements can be added. Conversely, given a leading subset, the corresponding antichain can be found by deleting all the elements that have successors in the subset. Thus the antichains are in one-to-one correspondence with leading subsets. Since the sizes of the antichains do not exceed K, there are no more than $(|V|+1)^K$ antichains, and so the number of leading subsets of V is bounded by a polynomial.

The leading subsets are partially ordered by set inclusion. If X is a leading subset and its corresponding antichain is the set $\{x_1, x_2, \ldots, x_k\}$, then the immediate predecessors of X are $X - \{x_1\}, X - \{x_2\}, \ldots, X - \{x_k\}$; we will denote the set of immediate predecessors of X by $\mathrm{Pred}(X)$.

To solve an instance of K-COLA, we use dynamic programming. There will be a subproblem for each leading subset of V. If X is a leading subset of V, let $B(X)$ be the badness of the best possible way to arrange X in the first $|X|$ pages of the embedding, where the badness contributed by the dangling edges is calculated as before. Then $B(X)$ can be calculated by the formula

$$B(X) = \min_{Y \in \mathrm{Pred}(X)} \{B(Y)\} + R(X),$$

for all nonempty leading subsets X, and with $B(\emptyset) = 0$. Here $R(X)$ is the sum of the weights on all the dangling edges:

$$R(X) = \sum_{\substack{(u,v) \in E \\ u \in X, v \in V - X \\ \text{or } v \in X, u \in V - X}} w((u,v)).$$

These values are calculated in topological order, so that when the badness of X is being calculated, the badness values for all of its leading subsets are already known. The value $B(V)$ calculated in this way is the smallest possible total badness for all linear arrangements of V that are consistent with the given partial order.

The K-COLA problem appears to be a non-trivial generalization of ML, and may even have applications in other areas, but for the purposes of pagination

we need to generalize in other ways. One generalization that is needed is to permit more than one item to be placed on a page; another is to allow glue between the text blocks. These things contribute to the badness function in a very localized way—we can account for them by defining a function that takes a subset of the items and a page number, and returns the badness of putting those items on the given page. This function would only need to apply to certain subsets of the items; for instance in the case where the partial order consists of a chain of text blocks and a chain of figures, a legal subset would contain a contiguous sequence of text blocks and a contiguous sequence of figures. We don't need to place any restriction on this component of the badness function, other than that it can be evaluated in a reasonable amount of time.

A DIFFERENT WAY TO GENERALIZE

Another way the algorithm should be generalized is to allow a more general way to specify the badness to be charged for figure placement. We have seen from the results of the previous chapter that this component cannot be a completely general function, but we would like to have an idea of how general it can be while still allowing a solution by dynamic programming. Ideally, we would like to have a necessary and sufficient condition on the figure-placement badness function that would say whether or not the function can be minimized by a dynamic programming algorithm. Finding sufficient conditions is no problem—we already have an example, in the K-COLA problem, of one of these. Proving that a condition is necessary, however, is a harder task. For one thing, the definition of 'dynamic programming' is sufficiently vague to make it difficult to pin down exactly—it is, after all, only a general approach for designing algorithms, not a rigorously defined mathematical notion. Another problem is that there are so many degrees of freedom present in the definition of the badness function that, given a simply stated criterion, it would probably be possible to cook up a function that failed to meet the criterion, but that still could be handled by an appropriate algorithm. In view of these difficulties, we will not attempt to find a necessary condition, but instead will aim for a sufficient condition that is not too complex to state, and yet encompasses a broad class of badness functions that will be useful in practice.

We will start by giving a very general way of specifying the figure placement component of the badness function, and then look for restrictions to place on it so that dynamic programming will still apply. Following this strategy, we have the

Generalized Pagination Problem (GP):

GIVEN: A partially ordered set $(V, \prec)$ of items;
a function $L(X, i)$ that gives the badness of putting the set $X \subseteq V$ on page i;
a function $R_{uv}(i, j)$ that gives the badness of putting $u \in V$ on page i and $v \in V$ on page j;
an integer p, the number of pages;
and a bound C.

QUESTION: Is there a mapping $P : V \to \{1, 2, \ldots, p\}$ such that $P(u) \leq P(v)$ whenever $u \preceq v$, and

$$\sum_{1 \leq i \leq |V|} L(P^{-1}(i), i) + \sum_{u,v \in V} R_{uv}(P(u), P(v)) \leq C \quad ?$$

The partial order $\prec$ is assumed to satisfy the same restriction as in the K-COLA problem, so that the number of leading subsets is bounded by a polynomial in the size of V. The mapping P is no longer required to be one-to-one, because we want to allow more than one item to go on a page. Now we would like to know what restrictions we need to place on the function R in order to be able to solve this problem by using dynamic programming.

The kind of algorithm we have in mind will calculate, for each leading subset X and each page number k, the value $B(X, k)$, which is the minimum badness for embedding X into the first k pages. Suppose we have embedded X into the first k pages, and $V - X$ into pages starting with $k + 1$. Let u be an element of X, and v an element of $V - X$. The principle of optimality says that if the badness cannot be decreased by changing the embedding of X or by changing the embedding of $V - X$, then it cannot be decreased by changing the embedding of both of these at the same time. The first part of the badness, computed from the function L, clearly satisfies this requirement. The second part of the badness will satisfy it also, if R_{uv} satisfies the condition

$$R_{uv}(i, j) - R_{uv}(i', j) = R_{uv}(i, j') - R_{uv}(i', j')$$

for $i, i' < j, j'$ and for $j, j' < i, i'$. This condition just says that the amount by which the figure placement badness changes when the position of u changes does not depend on where v happens to be, as long as they do not pass each other. This condition is the restriction we place on R_{uv} in order to guarantee that dynamic programming will work.

The restriction on R_{uv} can be restated in the form

$$\Delta_2\Delta_1 R(i,j) = 0 \qquad \text{for} \quad |i-j| > 1,$$

where Δ_k is the finite difference operator on the kth variable of the function:

$$\Delta_k f(x_1,\ldots,x_n) = f(x_1,\ldots,x_{k-1},x_k+1,x_{k+1},\ldots,x_n) - f(x_1,\ldots,x_n),$$

and where the subscripts have been dropped from R_{uv}.

Using this new form of the condition, we have

$$\begin{aligned} R(i,j) &= R(0,j) + \sum_{0\le k<i} \Delta_1 R(k,j) \\ &= R(0,j) + \sum_{0\le k<i} \left(\Delta_1 R(k,0) + \sum_{0\le k<j} \Delta_2\Delta_1 R(k,l) \right) \\ &= R(0,j) + R(i,0) - R(0,0) + \sum_{\substack{0\le k<i \\ 0\le l<j}} \Delta_2\Delta_1 R(k,l). \end{aligned}$$

The summand is zero in the region $k > l+1$, so in the case $i > j$, the range of the summation may be reduced to $0 \le k < j+1$, $0 \le l < j$. Thus, if $i > j$,

$$\begin{aligned} R(i,j) &= R(0,j) + R(i,0) - R(0,0) + \\ &\qquad R(j+1,j) - R(0,j) - R(j+1,0) + R(0,0) \\ &= R(i,0) - R(j+1,0) + R(j+1,j); \end{aligned}$$

and, similarly, if $i < j$,

$$R(i,j) = R(0,j) - R(0,i+1) + R(i,i+1).$$

Now $R(i,0)$, $R(0,i)$, $R(i+1,i)$, $R(i,i+1)$, and $R(i,i)$ are all functions of one variable that may be chosen independently of each other; we will write $f_1(i) = R(i,0)$, $f_3(j) = R(j+1,j) - R(j+1,0)$, $f_2(j) = R(0,j)$, $f_4(i) = R(i,i+1) - R(0,i+1)$, and $f_5(i) = R(i,i)$.

$$R(i,j) = \begin{cases} f_1(i) + f_3(j) & \text{if } i > j \\ f_2(j) + f_4(i) & \text{if } i < j \\ f_5(i) & \text{if } i = j \end{cases}$$

for arbitrary functions $f_1, f_2, \ldots, f_5$.

This kind of badness function is useful for practical applications, especially the portions contributed by f_1 and f_2. One way to think of this is that a contribution of $\Delta f_1(i+1)$ is made to the badness function whenever a reference from u to v passes over the boundary between pages i and $i+1$ in the reverse direction, and a contribution of $\Delta f_2(i+1)$ is made when a reference crosses the boundary between pages i and $i+1$ in the forward direction. Thus the pagination algorithm may charge a small penalty when a reference crosses a gutter, and a larger penalty when the page must be turned to get from a citation to its figure. Furthermore, the cost can depend on which direction the crossing is made, and the penalties can be different for every reference. The usefulness of f_3 and f_4 is not as clear, but they might be used, for instance, to force a figure to come after its citation by making f_3 infinite and f_4 zero. The function f_5 may just as well be incorporated into the local component, L, of the badness function, and so we will assume that f_5 is zero.

We can now solve GP by calculating $B(X,k)$, the badness of putting the leading subset X onto pages 1 thru k, according to the formula

$$B(X,k) = \min_{Y \subseteq X} \{B(Y,k-1) + L(X-Y,k) + D(Y,X,k)\},$$

where $D(Y,X,k)$ is the part of the badness due to dangling references crossing the boundary between pages k and $k+1$:

$$D(Y,X,k) = \sum_{\substack{u \in X-Y \\ v \in V-X}} (\Delta f_{vu1}(k+1) + \Delta f_{uv2}(k+1) + f_{vu4}(k) + f_{uv3}(k)).$$

The smallest possible badness for any embedding is then given by $B(V,p)$, and we need only compare this against C to answer the question. The actual embedding P that satisfies this bound may be extracted by methods similar to those used for the earlier problems.

Chapter 4

Implementation

Machines exist; let us exploit them
to create beauty—a modern beauty,
while we are about it. — A. Huxley

THE PREVIOUS CHAPTER developed the theory for an optimizing pagination algorithm, but hidden beneath the partial orders and the badness functions are many details that need to be brought to the surface when writing a practical pagination routine. The local badness function must be specified in a way that is easy to evaluate, because it will be evaluated many times. A practical algorithm has no need for the general partial order of GP—it will suffice to have a list of text and a list of figures. The routine will have to do some pruning to eliminate subproblems that have a high badness value, because even though the number of subproblems is polynomially bounded, there are still too many of them to solve in a reasonable amount of time. The order in which the subproblems are solved needs to be decided upon—the choice of this order makes a difference in how well the pruning techniques work and in the ways that memory needs to be accessed. These are the issues we will deal with in this chapter.

The model used by the practical routine is essentially as described in chapter one, but differs in some minor details. As described there, the figures were included as entries in the vertical list; here we will use two lists, one for the text and the other for the figures. The boxes in the text list will be allowed to contain citations to the figures; this will permit each figure to be referenced any number of times if this is desirable. The badness function for figure placement will not be quite as general as the one in the previous chapter; each reference will have a forward weight and a reverse weight, and each page boundary will have a weight value. Each time a reference crosses a page boundary, the contribution to the badness function will be the product of the page boundary weight with the forward or reverse reference weight, as appropriate.

The text list will be denoted as $t_1, t_2, \ldots, t_n$,
where $t_i = (\tau_i, h_i, d_i, y_i, z_i, p_i, f_i, u_i, v_i)$,

τ_i is the kind of item, either 'box' or 'penalty' or 'glue',
h_i is the height of the box or glue,
d_i is the depth of the box,
y_i is the stretchability of the glue,
z_i is the shrinkability of the glue,
p_i is the penalty to be charged for breaking at this item,
f_i is the figure cited by this box,
u_i is the forward reference weight for this citation,
v_i is the backward reference weight for this citation.

All nine of these values are never used in the same item, so when actually coding the routine we would use some kind of record whose contents were interpreted differently depending on the value of τ_i. However, for the purpose of describing the algorithm, it is convenient to call them by these names, and to assume that they are zero when the value is not relevant.

Since we are running short on letters, we will use capitals to denote the contents of the figure list $F_1, F_2, \ldots, F_m$:

$F_j = (K_j, H_j, D_j, Y_j, Z_j)$,
K_j is the kind of figure, either 'top' or 'bot',
H_j is the normal height of the figure,
D_j is the depth,
Y_j is the stretchability,
Z_j is the shrinkability.

It is useful to define t_0 and F_0 as

$$t_0 = (\text{'box'}, 0, 0, 0, 0, 0, 0, 0, 0) \quad \text{and} \quad F_0 = (\text{'top'}, 0, 0, 0, 0).$$

The abstract pagination problems discussed in the previous chapter had symmetry between the text items and the figures. This symmetry is not carried over into this implementation because of the presence of penalty items and glue in the text list. In the GP problem we allowed references between two text items, or between two figures. This feature is dropped in this chapter, not because it is very difficult to implement or because it would not be useful, but merely because we would like to avoid further complication in the notation.

Instead of keeping track of the subproblems in a huge table, we choose to remember them by using a list of *breaknodes*, one breaknode for each subproblem that has been found to have a feasible solution. This way we don't have to allocate any storage to remember the subproblems that have an unreasonably high badness, and we won't need to examine such infeasible subproblems when generating a new subproblem.

At any point in the algorithm, there will in general be some breaknodes that are no longer useful for generating new subproblems, because the total size of the material that would have to go onto a new page in order to extend the embedding is larger than the page size, even when the maximum shrinkability has been applied. These nodes are still needed, however, because some of them will be involved in the optimum solution, and we don't know until the end which ones these will be. Therefore the breaknodes are stored in two lists, the *active list* and the *passive list*; new breaknodes go into the active list, and are moved to the passive list after it becomes evident that they will no longer generate new subproblems. There is a certain danger with this approach that an item with a negative size or glue with a very large shrinkability might make it possible for a breaknode that has been moved to the passive list to generate a new subproblem. One case where this could happen is with the example of chapter one on page 14 where a sequence is defined that allows the bottom margin of the page to vary by an integral number of lines. The positive glue at the beginning may get partially or totally cancelled out by negative glue that comes later, but if the pagination routine didn't know about that possibility, it would go ahead and deactivate some breaknodes as soon as the positive glue had been examined. To avoid this problem we introduce the *safetymargin* parameter, and do not deactivate a breaknode until the compressed size of the text involved exceeds the page size by this amount.

In addition to the lists $t_1, \ldots, t_n$ and $F_1, \ldots, F_m$, the page breaking routine needs the following parameters:

- $b(k)$, the weight associated with the boundary between page k and page $k + 1$; typically this function will assume just two values, large for k even and small for k odd.
- *pageheight*, the height of the finished pages.
- *maxdepth*, the maximum allowed depth of the page. If the bottom box on the page has a depth that exceeds this value, it is raised so that its bottom edge is *maxdepth* units below the normal bottom baseline.

• *topbaseline*, the minimum height of the first line of text on the page.

The last three parameters could just as well be functions of the page number, or of the first text item on the page; we will assume that they are constant in order to simplify the presentation.

The algorithm as presented here will not look at *topbaseline*; instead, if any box in the text list had a height $h <$ *topbaseline*, we will assume that it has been replaced by a box of height *topbaseline* preceded by an infinite penalty item and glue with height h — *topbaseline*.

TEX allows the specification of a special insert item, called the *topsep*, that is to be inserted after the last topinsert on the page, if there are any. There is an analogous *botsep* for botinserts. We will not go into the details of how these affect the badness calculation, but the description of the algorithm will indicate where the appropriate adjustments need to be made.

To keep track of each subproblem and its best solution, each breaknode has the following fields, where a is the pointer to the breaknode:

text(a) = number of text items to put before the break;
figures(a) = number of figures to put before the break;
pages(a) = number of pages before the break;
badness(a) = lowest badness for breaking this way;
previous(a) = pointer to breaknode for previous page;
link(a) = pointer to next breaknode in list.

The following variables are precalculated for $0 \le k \le n$, $0 \le l \le m$:

$$\hat{h}_k = \sum_{1 \le i \le k} (h_i + d_i) \qquad \hat{H}_l = \sum_{1 \le j \le l} (H_j + D_j)$$

$$\hat{y}_k = \sum_{1 \le i \le k} y_i \qquad \hat{Y}_l = \sum_{1 \le j \le l} Y_j$$

$$\hat{z}_k = \sum_{1 \le i \le k} z_i \qquad \hat{Z}_l = \sum_{1 \le j \le l} Z_j$$

and we let *after*(k) be the smallest index $i > k$ such that either $i > n$ or τ_i is 'box'. These tabulated values allow the total height, stretch, and shrink on a page to be calculated quickly; for instance, if we need the total height and depth of all the text items from the first box after text item a up to text item b, we just do a subtraction:

$$\hat{h}_b - \hat{h}_{after(a)-1},$$

and similarly for the other values.

We are now ready for the top-level description of the algorithm:

```
begin ⟨tabulate the sums needed for the badness calculations⟩;
⟨create a breaknode corresponding to no text and no figures⟩;
i := 0;
while i ≤ n do
  begin if ⟨it is legal to break after t_i⟩ then
    begin ⟨deactivate useless breaknodes⟩;
    for j := 0 to m do
      ⟨find and record the best feasible ways to break after t_i and F_j⟩;
    end;
  i := i + 1;
  if i ≤ n then ⟨update totals to include t_i⟩;
  end;
⟨choose the best active node with n text items and m figures⟩;
⟨trace back to find the optimum breakpoint sequence⟩;
end.
```

The tabulation of the sums is straightforward, so we will omit its detailed description. The global variable A will be used to point to the first node in the active list, and P will point to the first node in the passive list. Thus

```
⟨create a breaknode corresponding to no text and no figures⟩=
  begin A := newnode(text = 0, figures = 0, pages = 0, badness = 0,
      previous = Λ, link = Λ);
  P := Λ;
  end.
```

Testing for a legal place to break is not hard to do:

```
⟨it is legal to break after t_i⟩=
  (τ_i = 'penalty') or ((τ_i = 'box') and (τ_{i+1} = 'glue'))
```

A breaknode will no longer generate new subproblems when the height of the text that would have to go on the new page, minus the shrinkability of the text, exceeds the pagesize by more than the safety margin; we don't need to adjust for *maxdepth*, since this would only make it harder for the text to fit on the page.

⟨deactivate useless breaknodes⟩=

```
begin a := A; olda := Λ;
while a ≠ Λ do
  begin i' := min(i, after(text(a)) − 1);
  minsize := (ĥ_i − ĥ_i' − d_i) − (ẑ_i − ẑ_i');
  if minsize > pageheight + safetymargin then
    ⟨remove node at a and move it to passive list⟩
  else begin olda := a; a := link(a);
    end
  end
end.
```

⟨remove node at a and move it to passive list⟩=

```
begin if olda = Λ then A := link(a)
else link(olda) := link(a) ;
newa := link(a); link(a) := P; P := a; a := newa;
end.
```

Now we have arrived at the most interesting part of the algorithm, namely finding and recording the new feasible breaks. There might be more than one way to break after t_i and F_j, each way ending up on a different page boundary. We would like to generate all of these on one pass through the active list; one way to do this would be to keep a table of new feasible breaks, indexed by page number. The entry for page k would record the best way known so far to break the text up to t_i and the figures up to F_j into k pages. After the whole active list had been examined, the new feasible breaks would be moved from the table into the active list.

But there is a better way. If the active list is always maintained in increasing order by page number, all the breaknodes that contribute to a given subproblem are grouped together in the list. Hence after all the breaknodes for page k have been processed, a new breaknode for page $k+1$ can be inserted at the right place in the active list before going on to process the old breaknodes for page $k+1$. This eliminates the need to build a table to keep new subproblems in, since we only need to work on one new subproblem at a time.

```
⟨find and record the best feasible ways to break after t_i and F_j⟩=
  begin ⟨calculate total weight of dangling references⟩;
  if dangweight ≤ danglimit then
    begin ⟨indicate no new feasible subproblem yet⟩; a := A;
    while a ≠ Λ do
      begin if j ≥ figures(a) then
        begin ⟨calculate adjustment ratio adjratio⟩;
        if (adjratio ≥ −1) and (adjratio ≤ adjratiolimit) then
          begin ⟨calculate total badness⟩;
          ⟨record best feasible subproblem⟩
          end
        end;
      if (link(a) = Λ) or (pages(link(a)) > pages(a)) then
        ⟨insert any new breaknode into the active list⟩;
      a := link(a);
      end
    end
  end.
```

This section of the code incorporates two mechanisms for pruning the infeasible subproblems; one limits the total weight of the dangling references, and the other limits the adjustment ratio. The latter mechanism is the same one used by TEX's line breaking algorithm, and guards against stretching the glue too much. The dangling reference limit keeps the algorithm from even considering such ridiculous things as putting all the figures on the first page when the citations come much later. We could imagine adding other types of pruning; for example a bound could be placed on the total badness of any subproblem.

It should be noted that, in unusual circumstances, this pruning could cause the discovered solution to be non-optimal—the optimal solution might violate one of these local bounds, but come out ahead because everything else works out perfectly, or nearly so. But by specifying these limits, the user is saying that any solution that violates them is too bad to consider, so what is really wanted is the best solution that satisfies the limits. It might also happen that

no such solution exists; in this case the active list becomes empty and the algorithm essentially gives up. A production version would probably have to do something more friendly for the user, as is done with the line breaking routine of TEX. In any case, the limits could be raised and the pagination retried, either automatically or manually.

Next we need to calculate the total weight of the dangling references. This comes in two parts,

$$forwardtotal(i,j) = \sum_{\substack{k\le i\\ l>j}} \left\langle \begin{matrix}\text{forward weight of reference}\\ \text{from text item } k \text{ to figure } l\end{matrix} \right\rangle, \text{ and}$$

$$backwardtotal(i,j) = \sum_{\substack{k>i\\ l\le j}} \left\langle \begin{matrix}\text{backward weight of reference}\\ \text{from text item } k \text{ to figure } l\end{matrix} \right\rangle.$$

These could be tabulated ahead of time, but it seems better to keep only the current row of each of these matrices, and update them when i increases; since these rows will only change when the text item is a citation, the total work done for the updating will be proportional to the number of figures times the number of citations. This is small compared with the amount of work we are doing in the rest of the algorithm.

Therefore, for the current value of i, let

$$ftot(j) = forwardtotal(i,j), \text{ and}$$
$$btot(j) = backwardtotal(i,j).$$

Initially, when $i = 0$,

$$ftot(j) = 0 \qquad \text{and} \qquad btot(j) = \sum_{k\mid f_k\le j} v_k.$$

Using these arrays, it is easy to expand

⟨ calculate total weight of dangling references ⟩=

$dangweight := ftot(j) + btot(j)$

The next few sections are straightforward to fill out:

⟨ indicate no new feasible subproblem yet ⟩=

begin *lowestbadness* := *infinity*; *bestprev* := Λ
end.

⟨ calculate adjustment ratio *adjratio* ⟩=

begin $i' := \min(i, after(text(a)) - 1)$; $j' := figures(a)$;
$height := (\hat{h}_i - \hat{h}_{i'}) + (\hat{H}_i - \hat{H}_{j'})$;
$numtopinserts := tottop(j) - tottop(j')$;
$numbotinserts := j - j' - numtopinserts$;
if $numtopinserts > 0$ **then** ⟨ account for topsep ⟩;
if $numbotinserts > 0$ **then**
  begin ⟨ account for botsep ⟩; $depth \leftarrow D_{lastbot(j)}$
  end
else if $i > text(a)$ **then** $depth \leftarrow d_i$
else $depth \leftarrow$ ⟨ depth of topsep ⟩ ;
if $depth > maxdepth$ **then** $depth \leftarrow maxdepth$;
$height \leftarrow height - depth$;
if $height \leq pageheight$ **then** $glue \leftarrow (\hat{y}_i - \hat{y}_{i'}) + (\hat{Y}_j - \hat{Y}_{j'})$
else $glue \leftarrow (\hat{z}_i - \hat{z}_{i'}) + (\hat{Z}_j - \hat{Z}_{j'})$;
if $glue < 0.0001$ **then** $glue \leftarrow 0.0001$;
$adjratio \leftarrow (pageheight - height)/glue$
end.

A few words of explaination are in order about this section of code. The complications arise from the practical need to align the bottom baselines on each page, whether the bottom box is a text item or an insert (typically a footnote). To aid in determining which one of these situations occurs, two additional precalculated arrays have been introduced:

$$tottop(j) = \text{number of topinserts among } F_1, F_2, \ldots, F_j, \text{ and}$$
$$lastbot(j) = \text{index of last botinsert among } F_1, F_2, \ldots, F_j.$$

If the glue is too close to zero or is negative, it is arbitrarily set to a small positive value; this gives a large adjustment ratio, as it should, unless the height is also very close to the page height. This is an example of the kind of details that arise when moving from theory to practice.

Continuing, we define

```
⟨calculate total badness⟩=
  totbadness := badness(a) + |adjratio|^3 + p_i + dangweight × b(pages(a) + 1)

⟨record best feasible subproblem⟩=
  if totbadness < lowestbadness then
    begin lowestbadness := totbadness; bestprev := a;
    end.

⟨insert any new breaknode into the active list⟩=
  if lowestbadness < infinity then
    begin
    newa := newnode(text = i, figures = j, pages = pages(bestprev) + 1,
        badness = lowestbadness, previous = bestprev, link = link(a));
    link(a) := newa; a := newa;
    ⟨indicate no new feasible subproblems yet⟩
    end.

⟨update totals to include t_i⟩=
  if f_i ≠ 0 then
    begin if u_i ≠ 0 then
      for k := 0 to f_i − 1 do ftot(k) := ftot(k) + u_i;
    if v_i ≠ 0 then
      for k := f_i to m do btot(k) := btot(k) − v_i;
    end.

⟨choose the best active node with n text items and m figures⟩=
  begin a := A; winner := Λ; lowestbadness := infinity;
  while a ≠ Λ do
    begin if (text(a) = n) and (figures(a) = m) and
          (badness(a) < lowestbadness) then
      begin winner := a; lowestbadness := badness(a);
      end;
    a := link(a);
    end
  end.
```

And finally,

```
⟨trace back to find the optimum breakpoint sequence⟩=
  begin if winner := Λ then
    ⟨there is no feasible solution⟩
  else begin a := winner;
    while a ≠ Λ do
      begin ⟨break after t_text(a) and F_figures(a)⟩; a := previous(a);
      end
    end
  end.
```

The implementation described in this chapter is, of course, only one of the many ways that the details of the pagination algorithm can be filled in. There are many variations possible, and we shall discuss some of them here. VARIATIONS

There is no limit to the number of variations that can be made on the local badness function, as long as it only depends upon what goes on a single page. It might, for example, take the page number into account, so that a particular piece of text, say a chapter heading, could be forced to an odd page. It could charge an extra penalty if the page contained more than one figure, or if it contained a citation but not the cited figure. The function might even invoke a heuristic page layout routine that tried to arrange the contents nicely on the page, and returned an appropriate badness; such a routine would need to be fast, though, because it would be called many times.

The existing algorithm is not very helpful when it fails to find a feasible solution; there should be some useful feedback so that the user can easily see where the input needs to be altered to allow a good solution.

Provided $b(k)$ is really a function of $k \bmod 2$, the routine does not really have to remember the best feasible break for each page number, but only the best break for an even page number and the best for an odd page number. This could be done by having two active lists, one for even page numbers and the other for odd page numbers. The page number would no longer need to be stored in the break node, because all that matters is its value modulo 2, and this could be deduced by knowing which active list contained it. This optimization could not be applied, however, if the page size were a function of the page number.

The line breaking algorithm of TEX incorporates some refinements to avoid making two adjacent lines with wildly different adjustment ratios. This same idea could be applied to the pagination algorithm to avoid a drastic change in the adjustment ratio between two facing pages.

Provided that the spacing between baselines was constant, the pagination routine could interact with the line breaking routine to gain some extra flexibility in the text. The line breaking routine would record in the vertical list the additional demerits for making the paragraph longer or shorter by a few lines; when the pagination routine calculated the badness, it would consider each of the possibilites and pick the one that gave the best results. This process is analogous to the selective use of abbreviations as an aid to line breaking, as discussed in the paper by Knuth and Plass [9].

If the baseline spacing is not constant, for example because of the presence of oversized characters in a line, the situation becomes more complicated. By using a simple reduction from the 2-partition problem, this pagination problem can be shown to be NP-complete. However, it is not strongly NP-complete, and can be solved in polynomial time if the heights of the lines are small integer multiples of some unit. For a more detailed discussion of strong NP-completeness, the reader is referred to the book by Garey and Johnson [12].

Chapter 5

Connections

Why should we look to the past in order to prepare for the future? Because there is nowhere else to look. — J. Burke

THIS FINAL CHAPTER is to point out the relationship of this thesis with earlier results, and to indicate some directions for further research.

EARLY COMPUTER AIDED TYPESETTING SYSTEMS

Michael Barnett was one of the first researchers to apply the power of a computer to typesetting. The photocomposition machines then in use accepted input that was encoded on paper tape, and Barnett realized that there was no reason why these tapes could not be produced by a suitably programmed computer. His group at M.I.T. began working on this problem in 1961, and developed a series of experimental typesetting systems culminating in the PC6 system [16]. This system, although quite impressive for its time, did not include features for footnotes or for automatic placement of figures. There were codes that allowed a section of text to be specified as not breakable; the system would set part of such a section below the normal bottom boundary of the page if a page break would ordinarily have fallen inside of it. PC6 was the forerunner of two distinct lines of typesetting systems, the RUNOFF/NROFF/TROFF family [17], and the PAGE-1/PAGE-2/PAGE-3 family [18]; the former group of systems were developed and used primarily in academic and research environments, while the latter systems found more commercial applications. None of these systems have comprehensive pagination routines built into them, but they provide primitives within a procedural language that may be used to specify how pagination is to take place. This makes it possible to write specifications for some quite complicated page formats, but no attempt is made to base the pagination decisions on more than local information.

About the same time as Barnett's work, a group headed by John Duncan pursued similar goals at the Computing Laboratory of the University of

Newcastle-Upon-Tyne. Duncan's experiments improved the state of knowledge about line breaking and hyphenation algorithms, and his survey work [19] points out many of the requirements that a typesetting system needs to satisfy before it can be useful in a commercial environment. He apparently did not attempt to apply the experience gained in developing the line breaking algorithms to the analogous pagination problem.

Some of the typesetting systems in use today are PUB, SCRIBE, BRAVO, and TEX [20,21,22,8]. These systems provide various degrees of flexibility in the specification of figures, footnotes, and displays, but all of them make the page breaking decisions one page at a time, based on local information. There are undoubtedly many other systems in use, especially in the "real world" of commercial printing, but it seems unlikely that they have pagination schemes that are much more elaborate.

FITTING IN WITH THE SYSTEM

The pagination methods developed in this thesis fit in more gracefully with some kinds of typesetting systems than others. Since it requires two passes, it fits in best with a two-pass batch-oriented system. Since the pagination does not need to be perfect in the initial drafts, the use of the optimizing algorithm should be optional, perhaps invoking a second pass in an otherwise one pass system. Another approach is similar to that taken by SCRIBE for cross-referencing: the information about the text and figure lists can be written as a separate file each time the system runs, and the page breaks can be chosen on the basis of the information recorded on the previous run. This means that in most cases the program would ultimately need to be run twice to get the correct pagination. A variation of this approach is to run the pagination routine as a separate program, taking the auxiliary file written by the first run of the typesetter as input, and producing another file that tells the typesetter how to do the pagination on the next run. This is the approach that was taken for the experimental versions of the pagination routine, using TEX as the host system.

PROCEDURAL VS. DECLARATIVE

People like to classify computer languages as being either *procedural* or *declarative*; a procedural language is one in which the specifications are written as instructions that the machine is to follow step-by-step in order to get the desired output. A declarative language, on the other hand, allows the user to specify goals and constraints that the output is to satisfy, and the system is to translate these into actions to produce an output that meets those goals and constraints. No real typesetting language can be placed

entirely into one of these categories or the other, but most of them have a predominant style of specification that can be used to classify them. Many typesetting systems appear to be declarative to the novice user, with the procedural specification of the 'styles' being left to experts.

The optimizing pagination algorithm fits in more naturally with the declarative style of specification. The use of glue, penalties and references is a non-procedural way of describing constraints.

BATCH VS. INTERACTIVE

Another way to classify systems is as *interactive* or *batch*. An interactive system lets the user see right away how the output is affected by changes to the document, and this instant feedback can be very helpful in maintaining the user's interest and avoiding the frustrations of waiting for the results of batch runs. A batch system processes the whole document, with little or no feedback to or guidance from the user. Each of these approaches has its advantages and disadvantages. The advantages of the interactive approach are fairly clear, but an interactive typesetting system requires special terminals that are not yet widely available, or standardized. The batch approach allows existing equipment to be used, with the possible exception of the output device, which may be shared by many users. The batch environment is somewhat more inclined to the notion of separating the task of the author from that of the compositor—the style of the output can be changed by merely changing to a different set of document design specifications. This is not to say that this cannot be done with an interactive system, but with such a system the author tends to take the appearance of the page into account as he writes, so that if the style is subsequently changed, it may not work out as well as the original style. This tendency is not necessarily a bad thing, however, because if the document design that the author uses while writing is the same as the one that will be used in the eventual publication, the result will likely be a better-looking and more readable document.

The optimizing pagination algorithm is designed for a batch environment, and so its application in an interactive environment is somewhat clumsy. The most promising immediate approach seems to be to have the interactive system work with a 'galley' of text not yet broken into pages. The pagination routine could then be applied when the user asks for the document to be printed. A more ambitious approach would be to modify the algorithm to fit in with the interactive environment; provided adequate storage is available, by using appropriate data structures it might be possible to update the

information in the breaknodes in response to small changes in the file without having to recompute them all. This kind of technique should be explored in connection with the line breaking problem before trying to make it work for pagination.

PROBLEMS FOR FURTHER RESEARCH

There are still many aspects of the pagination problem that are promising for further research. For example, alternatives to the global optimization approach should be investigated. One idea is to optimize locally, just taking the next few pages into account. It is not hard to see how to cook up examples that will lead such a locally optimizing algorithm astray, leading up to a bad break at the end, while causing the no-lookahead algorithm no problems. This shows that increasing the lookahead will not necessarily increase the quality of the output. However, it would be interesting to see how well such a bounded lookahead scheme would work in practice.

As mentioned in chapter four, the optimizing algorithm can be made to work in conjunction with heuristic page layout routines. There are other ways of incorporating approximations into the pagination routine; for instance, inset figures could be handled satisfactorily by doing the pagination as if they were full width figures with the same area as the original figures; afterwards the paragraphs could be re-broken in order to fit them around the figures. The optimizing line breaking algorithm would be helpful in this regard, since it could break the text on the page in such a way that it would come out to the right number of lines. Layout artists traditionally use approximate copyfitting techniques to do such tasks, so methods like this hold much promise.

Bibliography

[1] Douglas C. McMurtrie, *The Book: The Story of Printing and Bookmaking*, Oxford University Press, New York, 1943.

[2] Joseph Moxon, *Mechanick Exercises*, J. Moxon, London, 1683. Reprinted by the Typothetæ of New York, 1896, with preface and notes by T. L. De Vinne; also reprinted by Oxford University Press, London, 1958, and by Dover, New York, 1978. Quoted passages are from vol. 2, page 217.

[3] Theodore W. Chaundy, Percy R. Barrett, and Charles Batey, *The Printing of Mathematics,* Oxford University Press, London, 1957.

[4] Alexander Lawson, *Printing Types: An Introduction,* Beacon Press, Boston, 1971.

[5] Theodore Low De Vinne, *Correct Composition*, Vol. 2 of *The Practice of Typography*, Century, New York, 1901. The referenced material appears on pages 171–175 and 343.

[6] Bruce Rogers, *Paragraphs on Printing,* William E. Rudge's Sons, New York, 1943. Reprinted by Dover, New York, 1979. The quoted material appears on pages 50, 57, and 160.

[7] Paul E. Justus, 'There is more to typesetting than setting type,' *IEEE Transactions on Professional Communications* PC-15, 13–16, 18 (1972).

[8] Donald E. Knuth, *TEX and METAFONT: New Directions in Typesetting*, American Mathematical Society and Digital Press, Bedford, Massachusetts, 1979.

[9] Donald E. Knuth and Michael F. Plass, 'Breaking Paragraphs into Lines,' to appear in *Software: Practice and Experience* (1981); also available as Stanford CS report STAN-CS-80-828.

[10] Stephen A. Cook, 'The complexity of theorem-proving procedures,' *Procedings of the Third Annual ACM Symposium on Theory of Computing,* Association for Computing Machinery, New York, 151–158 (1971).

[11] Richard M. Karp, 'Reducibility among combinatorial problems,' in R. E. Miller and J. W. Thatcher (eds.), *Complexity of Computer Computations,* Plenum Press, New York, 85–103 (1972).

[12] Michael R. Garey and David S. Johnson, *Computers and Intractability,* W. H. Freeman, San Francisco, 1979.

[13] Michael R. Garey, David S. Johnson, and L. Stockmeyer, 'Some simplified NP-complete graph problems,' *Theoretical Computer Science* **1**, 237–267 (1976).

[14] Richard Bellman, *Dynamic Programming,* Princeton Univ. Press, Princeton, N.J., 1957.

[15] S. Even and Yossi Shiloach, 'NP-completeness of several arrangement problems,' Report No. 43, Department of Computer Science, Technion, Haifa, Israel, 1972.

[16] Michael P. Barnett, *Computer Typesetting: Experiments and Prospects,* M.I.T. Press, Cambridge, Mass., 1965.

[17] Joseph F. Ossanna, 'NROFF/TROFF User's Manual,' Bell Telephone Laboratories Internal memorandum, Murray Hill, New Jersey, 1975.

[18] John Pierson, *Computer Composition using PAGE–1,* Wiley-Interscience, New York, 1972.

[19] C. J. Duncan, 'Look! No hands!', *The Penrose Annual* **57**, 121–168 (1964).

[20] Larry Tesler, *PUB: The Document Compiler,* Stanford Artificial Intellegence Project Operating Note 70, 1973.

[21] Brian K. Reid and Janet H. Walker, *SCRIBE Introductory User's Manual,* Unilogic, Ltd., Pittsburgh, PA, 1980.

[22] *Bravo Manual,* XEROX Corporation, 1979.