On the Pagination of Complex Documents

Anne Brüggemann-Klein* Rolf Klein** Stefan Wohlfeil**

July 27, 1998

Abstract

The pagination problem of complex documents is in placing text and floating objects on pages in such a way that each object appears close to, but not before, its text reference. Current electronic formatting systems do not offer the pagination quality provided by human experts in traditional bookprinting.

One reason is that a good placement of text and floating objects cannot be achieved in a single pass over the input. We show that this approach works only in a very restricted document model; but in a realistic setting no online algorithm can approximate optimal pagination quality.

Globally computing an optimal pagination critically depends on the measure of quality used. Some measures are known to render the problem NP-hard, others cause unwanted side effects. We propose to use the total number of page turns necessary for reading the document and for looking up all referenced objects.

This objective function can be optimized by dynamic programming, in time proportional to the number of text blocks times the number of floating objects. Our implementation takes less than one minute for formatting a chapter of 30 pages. The results compare favorably with layouts obtained by Word, FrameMaker, or LaTeX, that were fine-tuned by expert users.

Categories and Subject Descriptors: H.4.1 [**Information Systems Applications**]: Office Automation—*Desktop Publishing*; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*Graphical User Interfaces*; I.7.2 [**Document and Text Processing**]: Document Preparation—*Typesetting*; F.1.2 [**Theory of Computation**]: Models of Computation—*Online computation*; G.1.6 [**Mathematics of Computing**]: Optimization—*Constraint optimization*;

1 Introduction

This work discusses the pagination problem of complex documents like scientific reports, journal or conference articles, books, or manuals. Such documents typically contain figures, tables, displayed equations, and footnotes that are *referenced* within the text. Lack of space or other reasons can prevent these objects from being placed at the very text positions where they are cited. Therefore, figures and other non-textual objects are usually allowed to *float* in the document, i.e. they may appear at a later position, even on a later page, than their first citations.

*Technische Universität München, Arcisstr. 21, D-80290 München, Germany.
email: brueggem@informatik.tu-muenchen.de

**FernUniversität Hagen, Praktische Informatik VI, Feithstr. 142, D-58084 Hagen, Germany.
email: rolf.klein@fernuni-hagen.de, stefan.wohlfeil@fernuni-hagen.de

This work was partially supported by the Deutsche Forschungsgemeinschaft, grant Br 1309/2-1.

On encountering a reference to a figure, the reader usually needs to look up the figure before reading on, in order to understand the exposition. To provide high reading efficiency, the figure should be positioned close to the reference, ideally on the same page.

The pagination problem is in distributing the document's contents on pages in such a way that each referenced floating object appears close to, but not before, its text reference. If an object is cited more often than once, it should be placed close to its first citation, according to Chi82 [1982]. At the same time, other criteria observed in classical bookprinting must be met. For example, pages should be close to one hundred percent full, no page should end with a single line of a new paragraph, etc.

One could be tempted to consider this problem somewhat antiquated. Are pages not just a passing state in the evolution from scrolls to scrollable windows? We do not think so. One reason is that reading from paper seems to be more efficient than reading from a computer screen. Hansen & Haas [1988] found that users presented with a text on screen were facing greater difficulties to get a *sense of text* than those furnished with a paper version, in particular in reading a text critically, or with text that is rather long or conceptually difficult. More recent studies show that paging is faster than scrolling [Dyson & Kipping, 1997, page 705] and that readers tend to recall better when reading in a page environment as compared to a scroll environment [Piolat *et al.*, 1997].

In order to place the document's contents on pages, paragraphs must be broken into lines. This *line breaking* problem has been studied by Knuth & Plass [1981]; it is handled superbly by systems like TEX. Therefore, we assume in this paper that the line breaking task is already finished when the pagination task starts. That is, the document's textual contents are given to us as a sequence of text lines whose identical length equals the fixed width of the page or the column*. Were it not for the floating objects, computing a pagination would now be easy: we would simply put the same number of lines on each page, maintaining identical base line distances.

With floating objects around, the pagination problem becomes complicated, as Plass [1981] has shown during the development of TEX. He found that the algorithmic complexity of computing an optimal pagination critically depends on the *badness function* used for measuring pagination quality. Some badness functions cause the problem to become NP-complete while others allow an optimal pagination to be found in polynomial time. We feel that these badness functions also differ in their relevance to practice and in the side effects they may cause, like e.g. under-full pages.

In the early 80th, main memory was typically too small to hold several pages at the same time. Thus, none of the global optimization methods discussed by Plass [1981] could be implemented in TEX. Instead, a simple first-fit pagination algorithm is employed until today in TEX as well as in LATEX [Lamport, 1986]. In a single pass, every figure is put on the first page after its citation where it fits. With this online approach, there is no need for dealing with more than one page at the same time. On the other hand, it is by no means clear how good this strategy works; to our knowledge, there are no previous theoretical results on this problem. But many users share the experience that ever so often LATEX's placement of figures can be greatly improved on by manual intervention. Users insert explicit page breaking commands into the input file, or they change the document's contents a little—a time consuming and tedious task that should better be avoided.

Since then the state of affairs in pagination has not changed very much. Asher [1990] built a new system Type & Set based on TEX as the formatting engine. Type & Set operates

*Note that this assumption does not hold for newspapers where figures are surrounded by text, so that the line length depends on the placement of floating objects.

in two passes. First, the text lines, figures and footnote lines are reconstructed from TEX's output file. Then, an optimizing pagination routine pastes the reconstructed material into pages. The measure to be optimized seems to mainly reflect page justification and balancing while figure placement is only a secondary concern. Kernighan & Wyk [1989] developed a page makeup program which postprocesses `troff` output. In order to keep the pagination algorithm as simple and efficient as possible, figure placement is also done in a single pass over the input.

The purpose of this paper is threefold. First, we investigate how good a pagination can be expected if the first-fit strategy for placing figures (that is used by current formatting systems) is used. The result is somewhat surprising. Whereas in the well-known *bin packing* problem the first-fit strategy is known to produce results at most seventy percent worse than optimal, no such guarantee can be given in the pagination problem. Only for a very restricted, unrealistic document model the first-fit strategy works well. Using competitive analysis we show that, under realistic assumptions, not only first-fit but *any* online pagination algorithm may produce results that are arbitrarily worse than necessary. This explains why so many people are not satisfied with paginations produced by LATEX if no manual improvement is done.

Second, we introduce a new quality measure for paginations that is aimed at high reading efficiency. We propose to count the total number of pages the reader needs to turn while reading through the document and looking up the references. Minimizing this number keeps the document short and places floating objects close to their references in the text. Either of these goals may be emphasized by introducing weights. An optimal pagination with respect to this quality measure can be computed offline, in time proportional to the number of text blocks times the number of floating objects, by a dynamic programming algorithm.

Since the relevance to practice of this approach cannot be theoretically argued, our third contribution is in reporting on the practical results we have obtained so far. We decided to implement a prototype called XFORMATTER that employs the above pagination algorithm. It deals with two types of floating objects: figures and footnotes.

XFORMATTER not only minimizes the number of page turns; without adding to the algorithmic complexity, it also satisfies a number of additional pagination constraints commonly used in professional printing. For example, it avoids generating *widows* and *orphans*, and it justifies all pages to the same height within a specified range. XFORMATTER can also be used on double-sided documents. Here the two pages on the same spread are adjusted to exactly the same height. A floating object may be positioned on a left page even though its reference appears only on the subsequent right page [Chi82, 1982]. Also, the user can specify that two figures should *not* appear on the same spread, a useful feature for exercises and solutions in textbooks.

We have run XFORMATTER on real-world documents that were previously formatted by Word, FrameMaker, or LATEX, and carefully fine-tuned by expert users. The results are very encouraging. Not only did the quality of paginations by XFORMATTER favorably compare with the previous layouts; even the run time needed to achieve this quality seems quite acceptable. Typically, it took us less than one minute to optimally format a chapter of some 30 pages, on a standard Sun workstation.

The rest of this paper is organized as follows. In Section 2 we formally specify the pagination problem. Also, we introduce a new measure for the quality of a pagination. Section 3 contains the competitive analysis of online pagination algorithms. In Section 4 we show that our measure of quality can be optimized by dynamic programming because the *principle of optimality* is satisfied. Section 5 contains our practical results. Finally, we summarize, and suggest some further research.

2 The Pagination Task

In this section we specify the types of documents studied in this paper, and we define input and desired output of the pagination task. In general, we adopt the notations of Furuta *et al.* [1982].

2.1 The Document Model

We assume that the underlying document model provides for several *streams* of logical objects. Within each stream, all objects are of the same type, like text, figures, footnotes, etc. The relative order of objects of the same stream must be preserved in the output model, i.e. it may not be changed by formatting the document.

Objects may refer to other objects of the same or of different streams; examples are cross references, bibliographical citations, footnotes, and references to figures or tables. For the pagination task the latter are of particular interest since they entail some freedom of choice. In fact, a footnote must start at the bottom of the same page that contains the reference to this footnote, but it may be continued at the bottom of subsequent pages. Figures and tables may float even more liberally; they may be printed on different pages than their referring text objects. An example document containing text, figure, and footnote streams is shown in Figure 1.

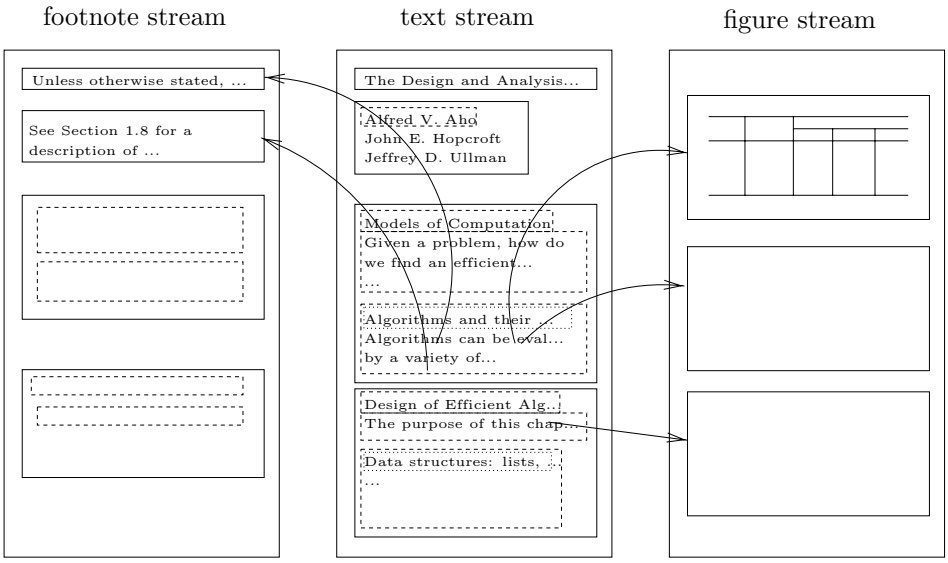

Figure 1: Logical structure of a document with three streams

As to the *text stream*, we assume that the line breaking task has already been completed. Afterwards, each paragraph consists of a sequence of line boxes and vertical glue boxes; compare Knuth & Plass [1981]; Plass & Knuth [1982]; Knuth [1986]. Figure 2 shows a typical example. All line boxes are of the same width, equal to the width of the text region. But their heights and depths may vary according to the line's textual contents. To make up for these differences, the vertical glue boxes between the lines (leading) usually have their heights so adjusted that all base line distances are the same; this regularity makes it easier to read the document. In general, each box is assigned a height and a depth as well as a capability to stretch or shrink that may depend on where, on a page, this box will appear, at the top, in the middle, or at the bottom.

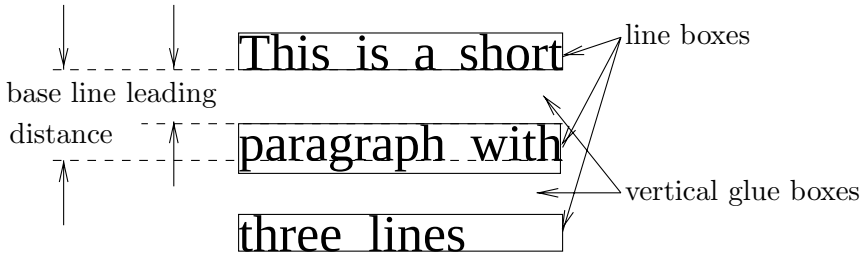

Figure 2: Result of line breaking task

Text boxes are denoted by t_i. We write $t_i < t_j$ if and only if t_i occurs earlier in the text stream than t_j does.

The *figure stream* is a sequence of boxes, too. The size of a figure box depends on the size of the figure itself and the size of its caption. Distances between figure boxes are modelled by vertical glue boxes, as in the text stream. We will denote the elements of the figure stream by f_i. Figure boxes have the same properties as line boxes.

While a figure can be referred to many times, the first citation matters most for the pagination task, since this is where the figure should ideally be placed, according to Chi82 [1982]. We write $R(f_j) = t_i$ to express that the first reference to figure f_j occurs in text box t_i; function R is called the *reference function*. We assume in this paper that the reference function is monotonic. That is, the figures appear in their figure stream in the same order as their first citations occur in the text: $f_i < f_j$ holds if and only if $R(f_i) < R(f_j)$ is fulfilled.

The *footnote stream* consists of text and glue boxes, n_i. It resembles the text stream in most aspects.

2.2 The Page Model

The basic properties of a page are its width and its height. We assume that each page contains but a single column. However, our approach can easily be generalized to pages with two or more columns of identical width. While discussing the potential of online pagination algorithms, in Section 3, we limit ourselves to single-sided documents, without loss of generality. Later, in Section 4, we show how our offline pagination algorithm can be applied to double-sided documents, too.

Each page contains a number ≥ 0 of *regions* for each stream. A region represents an area where material of the corresponding stream will be typeset, in a top-down manner. We assume that all pages of the document belong to the same *page class*. In this paper, two different classes will be discussed.

The *simple* page class consists of pages that contain an arbitrary number of text and figure regions, in any order. Regions are placed directly on top of each other, and the full page height is available; some example pages are shown in Figure 3. This page class will be used only in Section 3, in exploring the reach of online pagination algorithms.

In practice, one does not usually place a figure region directly on top of a text region; to better separate the figure's caption from the subsequent text, some extra white space d must be inserted between them. Moreover, it is quite standard, and greatly simplifies the pagination task, to position all figures on the top part of a page. This leads to our second, so-called *extra glue* page class. It consists of pages that either contain only one text region, or only one figure region, or a figure region on top of a text region, with white space d between them. Examples are shown in Figure 4.

In Section 3 we will see that the complexity of the pagination problem considerably increases by the need for extra glue between figures and text. Nonetheless, it is this more

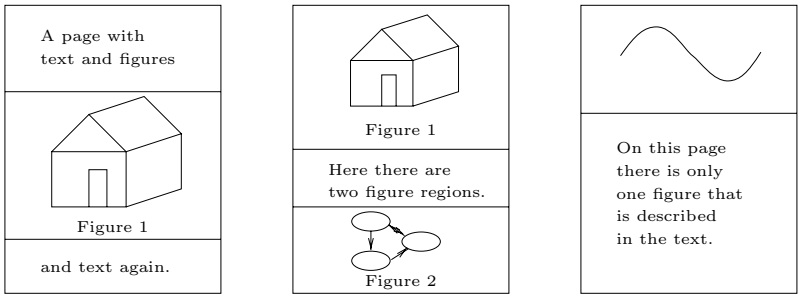

Figure 3: Some example pages of the *simple* page class

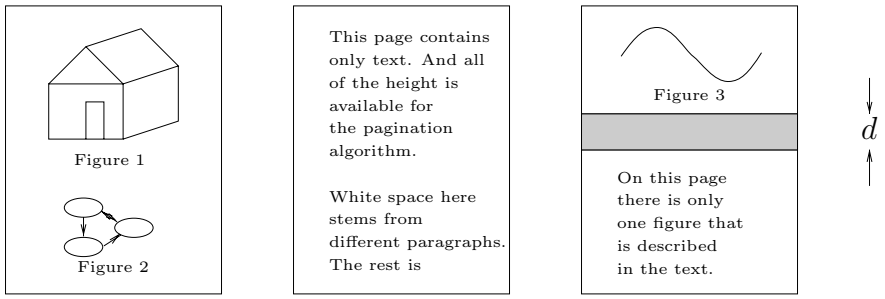

Figure 4: Some pages of the *extra glue* page class

realistic page class our offline algorithm is designed to deal with.

2.3 The Pagination Output

Formally, a pagination P would be a mapping from the input boxes $b_i \in \{t_i, f_i, n_i\}$ to a set of triplets (p, x, y); here p denotes the page number, and (x, y) are the coordinates where the reference point of the box is placed on page p.

$$P : \{b_1, \ldots, b_n\} \rightarrow (p, x, y).$$

In the sequel we are only interested in the number p of the page on which a box is placed; it will also be denoted by $P(b_i)$ as no confusion can arise.

2.4 The Pagination Goals

Pagination is difficult because a number of competing constraints and goals have to be satisfied simultaneously. In this section we will discuss these requirements. There are several style guides [Chi82, 1982; Williamson, 1983; Rubinstein, 1988] that describe the goals of the pagination process in more detail.

Sequence: The pagination task has to maintain the order of the boxes within their respective streams. (This is the main difference between the pagination task and the *bin packing* problem, where a number of objects have to be placed in bins but their order does not matter. How to use the smallest possible number of bins is a problem known to be NP-complete [Garey & Johnson, 1979], i.e. a polynomial time algorithm is not likely to exist.)

Widows and Orphans: The last line box of a paragraph which is set isolated at the top of a new page is called a *widow*. The first line box of a paragraph that is printed isolated at the bottom of a page is called an *orphan*. Widows and orphans may confuse the eye in

reading [Rubinstein, 1988] and may disturb a reader while parsing a page. Therefore, they should better be avoided. In the case of orphans, this can be achieved by disallowing the first line box of a paragraph to become the last box on a page.

Page Height: Each page should be perfectly full. This does not only look better than pages containing large areas of white space, it is also more economical because fewer pages are needed. Sometimes this may be too hard a constraint, especially if widows and orphans must be avoided, too. Therefore the page depth may be adjusted a little bit [Chi82, 1982], for example by allowing a page to run one line short. However, in documents printed on double-sided pages (like most books), the pages of one page spread must be equally full.

Floating Objects: Floating objects should be placed close to their *first* reference, but they must not be placed on a page before their citation. If a figure or table cannot be placed on the same page with its citation, it may be placed on a following page.

In documents that are printed on double-sided pages this constraint is relaxed. Here a floating object may be placed on the left-hand page while the citation is placed on the right-hand page. However, a floating object must not be placed on a page spread before its citation.

Grouping and Separating: Occasionally it is useful to place some objects on the same page or on the same page spread, for example two tables or figures that should be compared by the reader. On the other hand, there may be good reasons for *not* putting two objects on the same page or spread, for example an exercise and its solution in a textbook.

All requirements but one are rigorous; only the placement of floating objects can be done in many different ways. In the following section we discuss how to evaluate, and compare, paginations that differ in this respect.

2.5 Measuring the Pagination Quality

Text that is intended to be read continuously should have a high *legibility*. Experimental work cited by Rubinstein [1988] shows that legibility decreases whenever continuous reading is interrupted. Some interrupts occur at line ends, where the eye has to focus on the start of the next line.

If figures are not placed on the same page with their references, not only has the eye to focus on the figure, the reader even has to turn pages in order to locate the figure. Clearly, this interrupts continuous reading a lot more than re-focussing on the same page does. Therefore, Plass [1981] suggested to count the page differences between each figure and all of its references, as a measure of the badness of a pagination. He defined two badness functions,

$$\mathit{Lin}(P) = \sum_{i=1}^{n} \sum_{j=1}^{r(i)} \Big| P(f_i) - P(R_j(f_i)) \Big| \quad \text{and} \quad \mathit{Quad}(P) = \sum_{i=1}^{n} \sum_{j=1}^{r(i)} \Big(P(f_i) - P(R_j(f_i)) \Big)^2,$$

where $r(i)$ is the number of references to figure f_i from a line box, $P(b_i)$ denotes the number of the page where box b_i is placed, and $R_j(f_i)$ denotes the line box with the jth reference to f_i. Small values of $\mathit{Lin}(P)$ or $\mathit{Quad}(P)$ indicate a good pagination.

Plass proved that finding a pagination P, for an arbitrary given input, that minimizes *Quad* is an NP-complete problem, by transforming the well-known problem *maximum 2-satisfiability* to the pagination problem.

We see two problems with these badness functions.

1. They take into account all line boxes containing a reference to a given figure; but the Chicago Manual of Style [Chi82, 1982] requires to place floating objects close to their *first* referencing line box.

2. The quadratic badness function overvalues small improvements of very bad placements. Assume that pagination P_1 places 18 figures on the same page each with its reference, and one figure 10 pages behind its reference. Let us compare P_1 against a pagination P_2 of the same document, that places each of the 18 figures on the page after its reference and the last figure only 9 pages behind its reference. If measured with *Quad*, P_2 is better than P_1, because of

$$\mathit{Quad}(P_1) = 18 \cdot 0^2 + 1 \cdot 10^2 = 100 \quad \text{but} \quad \mathit{Quad}(P_2) = 18 \cdot 1^2 + 1 \cdot 9^2 = 99$$

   But obviously, pagination P_1 is preferable.

Therefore, we suggest to directly measure what really interrupts the reader: the total number of page turns that are necessary while reading the document.

There are two reasons for turning a page: (1) The reader has finished a page and continues with the next one. (2) The reader encounters a reference to a floating object which is not placed on the current page; in that case one has to turn to the next page, or to a page even further away, view the floating object, and flip back, before reading can be continued.

In formatting a document, these reasons may carry different weights. For a technical manual, for example, it could be important that all figures appear on the same page with their references, so that the manual can be held in one hand. Then the page turns of type (2) should be minimized, perhaps at the expense of some extra pages. On the other hand, for a conference paper it could be mandatory not to exceed a given page limit, so type (1) page turns should be minimized. For these reasons, we propose the following quality measure.

$$\mathit{Turn.S}(\alpha, \beta, P) = \beta(p-1) + \sum_{i=1}^{n} \alpha\Big(P(f_i) - P(R(f_i)) \Big).$$

Here, p denotes the total number of pages, and α, β are non-negative weights that add up to 1. Weight α applies to the effort of thumbing from a reference to its figure and back, and β to the regular progression from page to page. As no figure f_i is placed on a page $P(f_i)$ before its referencing line box, all terms in the sum are positive.

Choosing $\alpha = 1$ and $\beta = 0$, we get Plass's original goal function if there is only one reference to each figure.

This measure can be extended to double-sided pages, where page turns of type (1) occur only after reading every second page. Let $S(x)$ denote the index of the spread containing page x; usually $S(x) = (x \text{ div } 2) + 1$ holds. The measure for the quality of such a pagination is

$$\mathit{Turn.D}(\alpha, \beta, P) = \beta(S(p)-1) + \sum_{i=1}^{n} \alpha\Big(S(P(f_i)) - S(P(R(f_i))) \Big).$$

In principle, one could think of many ways to define a measure of pagination quality. But there is one requirement any sensible measure should fulfill.

Definition 1 A measure of pagination quality is called *natural* if it has the following property. Increasing the page (or page spread) difference between a single figure and its referencing line box, while leaving all other such differences unchanged, increases the pagination badness.

By this definition, the measures *Lin*, *Quad*, *Turn.S*, and *Turn.D* are all natural.

3 Online Pagination Algorithms

Most of today's document formatters like TEX, LATEX, `troff`, and pm use an *online* pagination algorithm that places all boxes on pages during a single pass over the input streams.

It is typical of an online algorithm not to know its whole input sequence in advance. Initially, it only knows the start of the input sequence; yet it has to act upon this incomplete knowledge immediately. Afterwards, the next input element is presented, and so on.

Clearly, an online algorithm A hampered by incomplete knowledge cannot perform as well as it could be if future inputs were known. By *competitive analysis* [Sleator & Tarjan, 1985; Fiat & Woeginger, 1996] its performance can be measured in the following way. Let I be an arbitrary input sequence. If I were fully known, the problem at hand could be solved optimally, incurring a minimal badness $C_{opt}(I)$. Now let $C_A(I)$ denote the badness caused by algorithm A on input sequence I. If there are constant numbers c and a such that, for each possible input sequence I,

$$C_A(I) \leq c * C_{opt}(I) + a$$

holds then algorithm A is said to be *competitive* with *competitive factor* c. Roughly, this means that A behaves at most c times worse than necessary, whatever the input may be[†].

The power of this approach can be demonstrated by the bin packing problem mentioned in Section 2.5. The first-fit algorithm puts every object into the first bin from the left where it fits. This simple online strategy is competitive with ratio $c = 1.7$, that is, it needs at most seventy percent more bins than necessary—a result all the more remarkable as computing the minimum number of bins is NP-hard; see Garey & Johnson [1979]!

In this section we investigate how well the first-fit approach, or more general online algorithms, can work for the pagination problem. For the sake of simplicity, we restrict ourselves to a very simple model. The documents are single-sided. They consist of one text and one figure stream; all glue boxes are of height 0. Each box may appear as the last one on a page. Since all pages are requested to be 100 percent full, a valid pagination need not always exist; a necessary condition is that the sum of the heights of all boxes is a multiple of the page height.

3.1 Where Figure First Fit Works Well

The *FigureFirstFit* algorithm proceeds by placing line boxes on the page until a reference to a figure is encountered. In this case the line box is placed, and the figure referred to is appended to a figure queue. Each time a line box has been placed onto a page, the figure queue is served. Serving the figure queue means to repeatedly place the first figure of the queue on the current page as long as there is enough room; if the current page is completely full, a new page is started. Otherwise, placing line boxes is resumed. Using a *first in first out* queue guarantees that the figures are placed in the same order of their references.

Clearly, this algorithm works without lookahead. Once a page has been filled it is never changed, no matter what input will be read in the future.

We are now going to show that this simple approach does, in fact, work well in a very restricted document model. In addition to the assumptions made above, we assume the following. Text and figure boxes can neither stretch nor shrink; the line box height is fixed, and the page height, as well as all figure heights, are multiples thereof; pages are of the *simple* page class type introduced in Section 2.2. Under these assumptions we can prove the following.

[†]Note that we are addressing the quality an online algorithm achieves, not its running time consumption.

THEOREM 2 *If a pagination for a given input exists, then the* FigureFirstFit *algorithm is guaranteed to find a pagination. Under each natural measure of pagination quality, the pagination found is optimal.*

PROOF. In this paper we only sketch the rather technical proof; details can be found in Wohlfeil [1997]. In order to show that the algorithm finds a pagination whenever there is one, we prove that any valid pagination can be transformed into the pagination constructed by *FigureFirstFit*. This can be accomplished by simply swapping a figure with a single preceding line box on the same page, or with a number of line boxes at the bottom of the preceding page whose total height equals the height of the figure, provided that the reference to the figure is not overrun.

To prove that a pagination compiled by *FigureFirstFit* is optimal with respect to any natural measure, we first show that each figure is placed, by *FigureFirstFit*, as close to the beginning of the document as possible. Hence, no figure can be closer to its referencing line box, not even at the sacrifice of others, in any valid pagination. Now let M be an arbitrary natural measure of pagination badness, and let P be any valid pagination of the input document. Since in P every figure is at least as far away from its citation as in the pagination constructed by *FigureFirstFit*, the badness of P under M can only be larger, because M is natural; compare Definition 1. □

In terms of competitive analysis, Theorem 2 states that *FigureFirstFit* is competitive with (optimum!) competitive factor 1, in the restricted document model introduced in this section.

At least one of the restrictive assumptions on the document model can be relaxed while still maintaining Theorem 2: We can allow for line boxes of different heights. In this more general situation the *FigureFirstFit* algorithm, as introduced above, sometimes fails to find a pagination even though one exists. But this problem can be overcome by carefully adding a backtracking mechanism; see Wohlfeil [1997] for details.

However, other restrictions on the document model cannot be dropped as the following section shows.

3.2 Where No Online Pagination Algorithm Works Well

In this section we first investigate what happens if the simple page class is replaced with the more realistic *extra glue* page class introduced in Section 2.2. That is, a page may now contain text only, or figures only, or a figure region on top of a text region, separated by some extra glue, d. Otherwise, we adhere to the assumptions made in Section 3.1. Rather than arbitrary natural badness measures we are now using

$$C_A(I) = \sum_{i=1}^{n} P(R(f_i)) - P(f_i);$$

this expression equals the badness function *Turn.S* introduced in Section 2.5 if we choose $\alpha = 1$ and $\beta = 0$.

Technically, it would be possible to adapt the *FigureFirstFit* algorithm to the extra glue page class. But there is little to gain by this approach, as the following surprising result shows.

THEOREM 3 *In the above model, no online algorithm for the pagination task is competitive.*

PROOF. Let A be an online algorithm for pagination. Now consider the following pagination input. The text stream consists of $n + 16$ line boxes $\{t_1, \ldots, t_{n+16}\}$ that are of the same

height, 1. The figure stream contains $n+2$ figures $\{f_1, \ldots, f_{n+2}\}$ whose heights are as follows:

figure	f_1	f_2	f_3	$\ldots$	f_n	f_{n+1}	f_{n+2}
height	1	1	3		3	1	1

The page height equals 5, and the extra glue between the figure region and the text region is of height 1.

When algorithm A starts, it sees a figure f_i just at the moment it processes its referencing text line, $R(f_i)$. Figures f_1 and f_2 are referred to in the first two line boxes, that is $R(f_1) = t_1$ and $R(f_2) = t_2$. Up to line box t_7 there are no more references to figures. Now algorithm A has essentially[‡] two choices in filling the first two pages as shown in Figure 5.

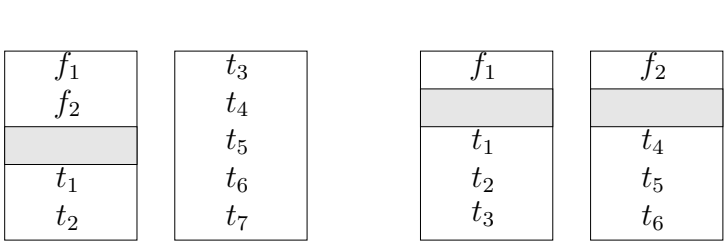

Figure 5: Two choices of an online pagination algorithm

The idea of the proof is as follows. No matter which of the two alternatives algorithm A chooses, we can always continue the input streams in such a way that the resulting pagination becomes very bad. At the same time, there exists a different pagination of the same input that is tremendously better than what A has achieved.

1. If algorithm A chooses alternative 1, the referencing line boxes of $f_3, \ldots, f_{n+2}$ will be as follows

figure	f_3	f_4	$\ldots$	f_i	$\ldots$	f_n	f_{n+1}	f_{n+2}
referencing line box	t_{12}	t_{13}		t_{i+9}		t_{n+9}	t_{n+10}	t_{n+14}

Now algorithm A has only the way shown in Figure 6 to complete the pagination.

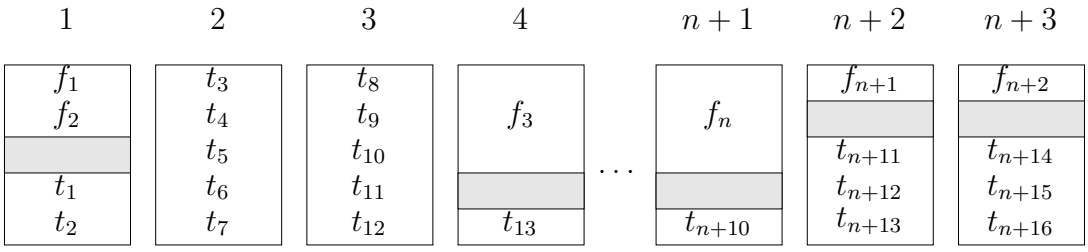

Figure 6: Pagination choice 1 completed

Indeed, algorithm A cannot find a better pagination, because (1) figure f_3 cannot be placed on page 3, because in that case f_3 would be placed before its reference t_{12}, (2) each of the figures $f_3, \ldots, f_n$ needs a page of its own, (3) the remaining space on

[‡]The algorithm could decide on not placing any figure on page 1 etc. But the resulting paginations would be even worse than the ones discussed.

pages $4, \ldots, n+1$ therefore must be filled with one line box each, and (4) the last two pages are already optimally filled and f_{n+2} is on the same page as its reference.

The badness of this pagination is $n-1$ because f_1, f_2, and f_{n+2} are on the same page as their references and the remaining $n-1$ figures are one page after their references. On the other hand the pagination shown in Figure 7 is also valid—and much better!

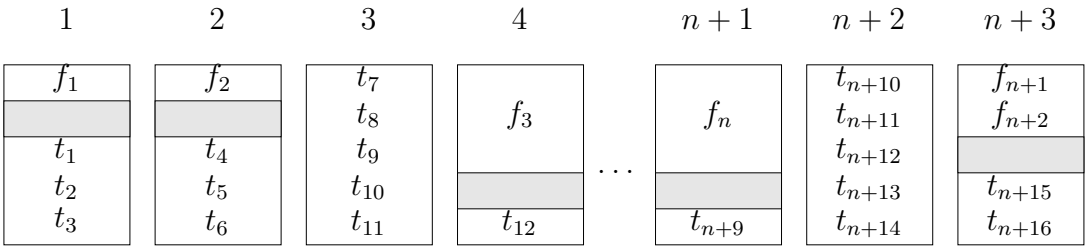

Figure 7: Optimal pagination for the given input

In this pagination only f_2, f_{n+1}, and f_{n+2} are placed one page after their referencing line box; all other figures are optimally placed. Thus, a badness of 3 results.

2. If algorithm A chooses alternative 2, the referencing line boxes of $f_3, \ldots, f_{n+2}$ will be the following

figure	f_3	f_4	$\ldots$	f_i	$\ldots$	f_n	f_{n+1}	f_{n+2}
referencing line box	t_8	t_9		t_{i+5}		t_{n+5}	t_{n+8}	t_{n+9}

Again, there is only one way for the online algorithm to continue the pagination (see Figure 8).

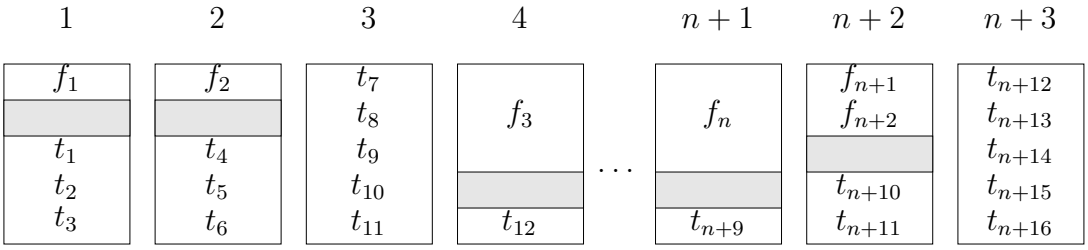

Figure 8: Pagination choice 2 completed

It is impossible for algorithm A to place f_3 on page three, because $R(f_3) = t_8$ would not also fit on page three. As no two of the figures $f_3, \ldots, f_n$ fit on the same page, there is only one way to fill the pages up to page $n+1$. Figures f_{n+1} and f_{n+2} are already referred to when page $n+1$ is completed, so it is optimal to place them immediately on page $n+2$. They cannot be placed on page $n+1$ because figure f_{n+2} would then be one page before its reference t_{n+9}. The page differences between a figure and its referencing line box are:

figure	f_1	f_2	f_3	f_4	f_5	f_6	$\ldots$	f_n	f_{n+1}	f_{n+2}
page difference	0	1	1	2	3	4		4	2	1

So the badness of this pagination is $4(n-5)+10 = 4(n-3)+2$. On the other hand the pagination in Figure 9 is also a valid pagination of the same input, but its badness is zero since all figures are placed on the same page with their referencing line boxes.

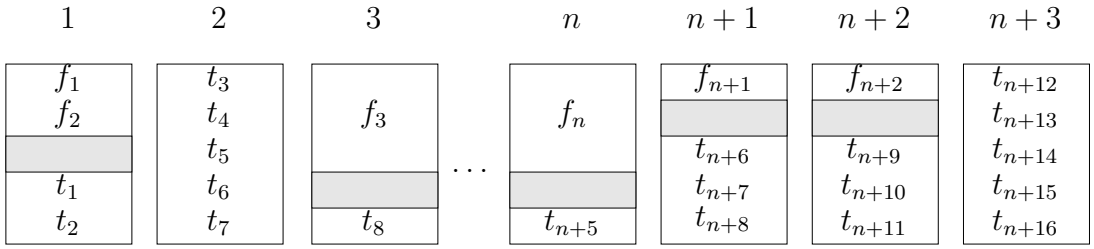

Figure 9: Optimal pagination for the given input

No matter which case applies, the pagination constructed by algorithm A is of badness at least $n-1$ whereas the optimal pagination incurs a badness of at most 3. Since the value of n can be arbitrarily large, algorithm A is not competitive. □

The proof of Theorem 3 reveals another remarkable fact. Even if the pagination algorithm were allowed to inspect the input streams in advance and to check the sizes of all boxes, it could still not be competitive without knowing which figure is referred to by which line box, i.e. without knowing the reference function, R!

In the above construction, the differences in pages between the figures and their citations are at most 4. One might wonder if there exists a general upper bound B with the following property. For each possible input there is a pagination P such that each figure is at most B pages further away from its citation as it would in a pagination that places this particular figure as close to its citation as possible. The answer is negative, as has been shown in Wohlfeil [1997].

In Theorem 3 we have seen that online pagination algorithms do no longer work well if we replace the simple page class with the more realistic *extra glue* page class. The same happens if we adhere to the simple page class but relax the assumption that line boxes cannot stretch or shrink.

THEOREM 4 *Given a document class as above, but with line boxes that may stretch or shrink, and pages of the simple page class. In this model no online algorithm is competitive.*

The proof is similar to the proof of Theorem 3; for details see Wohlfeil [1997].

4 Pagination by Dynamic Programming

In the preceding section we have seen that high pagination quality cannot be achieved in a single pass over the input. Therefore, we resort to offline methods, where all input streams are completely read before the pagination algorithm starts. Clearly, this approach cannot be as time and space efficient as online algorithms are. On the other hand, it makes it possible to obtain a truly optimal pagination, provided that the optimization problem is computationally tractable.

In Section 2.5 we have mentioned that the computational complexity of finding an optimal pagination depends on the underlying badness function. The main goal of this section is in proving that the measures *Turn.S* and *Turn.D* introduced in Section 2.5 do allow for efficient optimization, and to provide a suitable algorithm.

To this end, we are using an algorithmic technique known as *dynamic programming*; see Bellman [1957]; Cormen *et al.* [1990]. The same technique has been successfully employed in the line breaking task; see Knuth & Plass [1981]. For dynamic programming to work the *principle of optimality* must hold: The optimal solution of the main problem consists

of optimal solutions of subproblems. This fact enables building up an optimal solution in a bottom-up way, by combining the optimal solutions of subproblems of ever increasing size.

We are working with the *extra glue* page class introduced in Section 2.2. First, we discuss single-sided documents; then we show how to extend our algorithm to double-sided pages.

4.1 Pagination for Single-Sided Pages

Suppose we are given a text stream of line boxes $t_1, \ldots, t_m$, and a figure stream $f_1, \ldots, f_n$. Let us assume that, for some indices k, l, we have already found a *partial pagination* $P_{k,l}$ that neatly places the line boxes $t_1, \ldots, t_k$ and figure boxes $f_1, \ldots, f_l$ onto p pages, in accordance with the pagination goals stated in Section 2.4. Such a partial pagination differs from a complete one in that there may be dangling references: Some textblock t_h, where $h \leq k$, may contain a reference to a figure f_w, $l < w$, that has not been put on a page yet.

We adapt our measure *Turn.S* to also include dangling references in the following way. A figure f_w referred to in text box t_h, $h \leq k$, that does not appear on one of the p pages of partial pagination $P_{k,l}$ gives rise to the additive badness term

$$p + 1 - P_{k,l}(R(f_w)),$$

multiplied by the weight factor α. That is, instead of counting how many pages separate the citation from the figure—whose position is not yet known!— we charge the least number of page turns this figure is going to cause, namely if it were positioned on the very next page. With this extension, we can speak about the badness,

$$\mathit{Turn.S}(\alpha, \beta, P_{k,l}),$$

of a partial pagination $P_{k,l}$.

Note that for arbitrary values of k and l a partial pagination $P_{k,l}$ need not exist, because it may be impossible to completely fill a number of pages with the text and figure boxes given while complying with the pagination rules. But let us assume that it does, and that $P_{k,l}$ is optimal, with respect to the (extended) measure *Turn.S*, among all paginations of the first k line boxes and the first l figure boxes. If we remove the last page from $P_{k,l}$ we obtain a partial pagination $P_{i,j}$, for some indices $i \leq k$ and $j \leq l$. Now the following fact is crucial.

THEOREM 5 *If partial pagination $P_{k,l}$ is optimal with respect to measure* Turn.S *then the partial pagination $P_{i,j}$ that results from removing the last page of $P_{k,l}$ is optimal, too.*

PROOF. Let us consider how the values of $\mathit{Turn.S}(\alpha, \beta, P_{k,l})$ and $\mathit{Turn.S}(\alpha, \beta, P_{i,j})$ differ. Since there is one page more to turn in $P_{k,l}$, its badness has an extra additive term $\beta * 1$; compare the definition in Section 2.5. Suppose $P_{k,l}$ involves p pages. Each figure placed on the first $p - 1$ pages contributes identically to either badness value. The same holds for each figure possibly placed on page p, according to how $\mathit{Turn.S}(\alpha, \beta, P_{i,j})$ charges a dangling reference.

Next, let us assume that a figure f_w is cited, but not placed, in $P_{k,l}$. Then its contribution to the badness of $P_{k,l}$ is larger by $\alpha * 1$ because this pagination exceeds $P_{i,j}$ by one page. The same argument holds if the citation of f_w occurs on page p. Consequently, we have

$$\mathit{Turn.S}(\alpha, \beta, P_{k,l}) = \mathit{Turn.S}(\alpha, \beta, P_{i,j}) + \beta + \alpha * q(k, l),$$

where $q(k, l)$ denotes the number of those figures of index $> l$ whose citation occurs within a text box of index $\leq k$. One should observe that this quantity does not depend on the pagination $P_{k,l}$ itself but only on the indices k, l and on the input streams.

Now the claim of the theorem is immediate. If there were a partial pagination $P'_{i,j}$ of the same substreams $t_1, \ldots, t_i$ and $f_1, \ldots, f_j$ with a badness smaller than that of $P_{i,j}$ we could simply substitute it, and obtain a new pagination $P'_{k,l}$ whose badness would in turn be smaller than that of $P_{k,l}$, by the above formula. But this would contradict the assumed optimality of $P_{k,l}$. □

One should observe that this proof would break down for a quadratic badness measure like *Quad* discussed in Section 2.5. See also Plass [1981].

The concept of partial paginations is illustrated by Figure 10. It refers to an example document containing 647 text lines and 18 figures. An asterisk at position (i, j) indicates that a partial pagination $P_{i,j}$ exists.

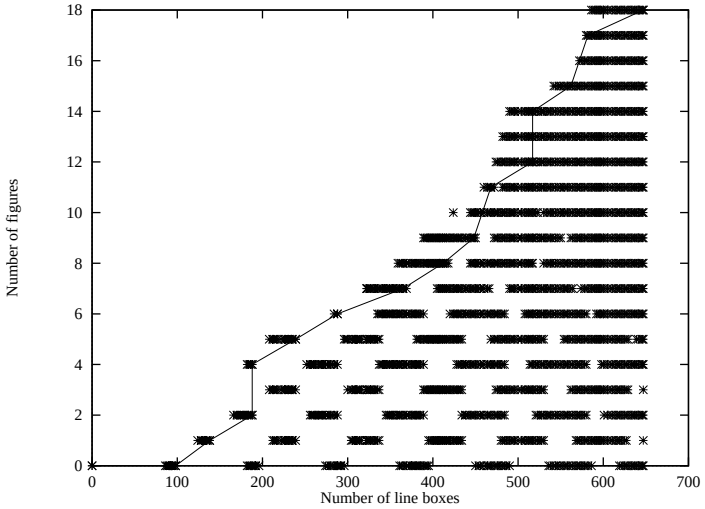

Figure 10: All partial paginations $P_{i,j}$ of an example document. An asterisk appears at position (i, j) if $P_{i,j}$ exists.

The top-left part in Figure 10 is empty because no figure can appear before its referring line box. But as figures may be placed many pages after their citations, the bottom-right part of the figure is filled with paginations. The row of asterisks near position $(92, 0)$ shows that there is more than one way to fill the first page, due to stretch- and shrinkability. In fact, if all line boxes are stretched to their limits the first page can be filled with only 86 line boxes, while shrinking produces a page with 98 line boxes. The zig-zagged line from the top-right corner $(647, 18)$ to the bottom-left corner $(0, 0)$ connects those optimal partial paginations that contribute to the optimal global solution.

In order to construct optimal paginations, we apply the dynamic programming technique in the following way. Suppose we want to determine if there exists a valid partial pagination $P_{k,l}$, and if so, compute an optimal one. At this time, all optimal partial paginations $P_{i,j}$, where $i \leq k, j \leq l$ and at least one index is strictly smaller, are already known. We check for which of them it is possible to place the remaining boxes $t_{i+1}, \ldots, t_k$ and $f_{j+1}, \ldots, f_l$ on a single page without violating the pagination goals. Whenever this can be done we compute the badness of the resulting pagination $P_{k,l}$. Among all candidates $P_{k,l}$ hereby obtained, the best one is chosen, and stored for further use. Theorem 5 ensures that by this way we really obtain an optimal partial pagination $P_{k,l}$, if such a pagination exists. By proceeding with ever increasing values of k and l, we finally arrive at an optimal pagination $P = P_{m,n}$ of the whole document[§].

[§] Or we learn that no pagination of the document is possible that does not violate the pagination goals; in this case human intervention is required, as will be discussed later.

The asymptotic running time of this algorithm can be estimated as follows.

THEOREM 6 *An optimal pagination of a document containing m text boxes and n line boxes can be compiled in time $O(nm)$.*

PROOF. At first glance it seems as if the above algorithm had, for each pair of indices (k, l), to check all partial paginations with a smaller index pair (i, j). As these are quadratic in number, an overall running time in $O(n^2m^2)$ would result. But a lot of this work can be saved. Because there is a lower bound to the possible height of any text or figure box, at most a certain number of them—say w—can fit onto one page. Therefore, we need to check only those optimal partial paginations $P_{i,j}$ where $m-w \leq i \leq m$ and $n-w \leq j \leq n$ hold—a constant number! As such checks are carried out for each index pair (k, l), the claim follows. □

The naive implementation of our pagination algorithm would use a $m \times n$ array for storing the relevant values of all optimal subpaginations $P_{i,j}$. How to save on the storage space needed while, at the same time, maintaining good performance has been explained in Wohlfeil [1997].

4.2 Pagination for Double-Sided Pages

With double-sided documents some careful modifications are necessary because we have two types of pages now, left ones and right ones, that must be treated differently.

When extending our badness measure *Turn.D* introduced in Section 2.5 to a partial pagination of substreams $t_1, \ldots, t_i$ and $f_1, \ldots, f_j$ we have to take into account the type of the last page. If it is of type left then all dangling references are ignored, for they may still be harmless if the corresponding figures are placed on the right page immediately following. But if the partial pagination ends with a right page, dangling references count as in the case of single-sided documents.

One should observe that we can no longer talk of *the* optimal partial pagination $P_{i,j}$ because there may be two of them, ending on a left and on a right page, correspondingly. Both of them must be considered by our dynamic programming algorithm.

Apart from these technical differences, the same results can be achieved as in Section 4.1.

4.3 Additional Constraints

The constraint that all pages have to be exactly filled is often so hard that no valid pagination can be found. Book designers then sometimes decide to have one or more under-full pages. To implement this capability, only *one* minor change has to be made. The user may specify that the sum of the heights of the material on one page has to be at least a given percentage of the page height. In other words, the actual page height is allowed to take its value in an interval like $[0.9 \cdot \mathit{pageHeight}, \mathit{pageHeight}]$. However, if double-sided pages are available the pages of one page spread have to be balanced.

A user may wish to further constrain the pagination task. According to Rubinstein [1988] there are four kinds of grouping and separating constraints:

same-page: two boxes are to be placed on the same page; the boxes may be from different streams or from the same stream.

different-page: two boxes must be placed on different pages.

same-spread: two boxes must appear on the same page spread.

different-spread: two boxes are to be placed on different page spreads.

The need for such constraints usually arises when a user is not satisfied with a pagination automatically produced by the system and wants to specify some additional constraints before repeating the pagination task. Constraints like these are best specified interactively. More details about how to algorithmically deal with them and about our prototype implementation in general may be found in Brüggemann-Klein *et al.* [1996]; Wohlfeil [1997].

5 Practical Results

5.1 Pagination Quality

Measuring the time performance of an algorithm in terms of "Big-O" alone does not tell very much about its relevance to practice. Also, it is not clear how much better an "optimal" pagination really is, as compared to one compiled by e.g. LaTeX, without actually computing the two, and comparing them directly. To this end we have implemented our pagination algorithm and run some experiments on different real-world documents. In this section we present some of the results.

The documents we are reporting on have originally been produced with LaTeX, FrameMaker, or Microsoft Word. They were fine-tuned by expert users, with the intention of improving on the pagination quality. The first document, *doc1*, is a report for the government of Bavaria [EFI95, 1995]; *doc2* is a chapter of a text book on Software Engineering [Pagel & Six, 1994]. The third document, *doc3*, is part of a Ph. D. thesis on school development research [Kanders, 1996] in Germany. The last document, *doc4*, is a chapter of a book on Computational Geometry [Klein, 1997].

Table 1 compares the quality of a first-fit pagination, a *Turn.S*-optimal pagination, a *Turn.S*-optimal pagination where pages need not be exactly filled but may be only 90% full, and the pagination produced by the authors. This table shows that the quality of first-fit

	doc1	doc2	doc3	doc4
Author's pagination	30	72	–	72
first-fit pagination	39	70	55	56
Turn.S-optimal pagination	31	65	47	52
Turn.S-90% optimal pagination	–	59	38	47

Table 1: Badness of single-sided paginations measured by *Turn.S*

algorithms is rather unpredictable. In fact, it may be quite well or it may be very poor, depending on the document's contents. On the other hand, *Turn.S* optimal paginations are usually of a quality comparable to the author's pagination. But if we allow pages to remain a little under-full, the optimizing algorithm is able to find even better paginations.

If some pages are not 100% full, some lines of text or some figures have to be moved from the under-full page to a following page. This may lead to a pagination which needs more pages than a pagination which completely fills all pages. However, if the last page was very loosely filled before (say with only a few lines), the optimization may be possible without using another page; the last page just gets more material. Our examples have shown that a significant improvement may be achieved if only *one* more page is used.

Paginations computed for single-sided pages may be printed on double-sided pages and vice versa. But a *Turn.D*-optimal pagination need not be *Turn.S*-optimal if printed on single-sided pages; it may even be invalid because a figure might be placed on a page before

its reference! Conversely, *Turn.S*-optimal paginations are not *Turn.D*-optimal if printed on double-sided pages. Therefore, it is essential for the pagination algorithm to know on what kind of paper the document will be printed. The following table shows which improvement in quality was possible, with our example documents, by using double-sided pages, and by optimizing accordingly.

	doc1	doc2	doc3	doc4
Author's pagination	13	37	–	39
first-fit pagination	17	38	25	26
Turn.D-optimal pagination	13	30	25	25
Turn.D-90% optimal pagination	10	26	18	23

Table 2: Badness of double-sided paginations

Again we can see that the quality of the first-fit paginations varies a lot. Allowing pages to be only at least 90% full (but have pages of one page spread balanced!) has produced the best results. These paginations are substantially better than what the authors had been able to achieve manually. In our examples, it was not necessary to append more than one extra page to get the best results.

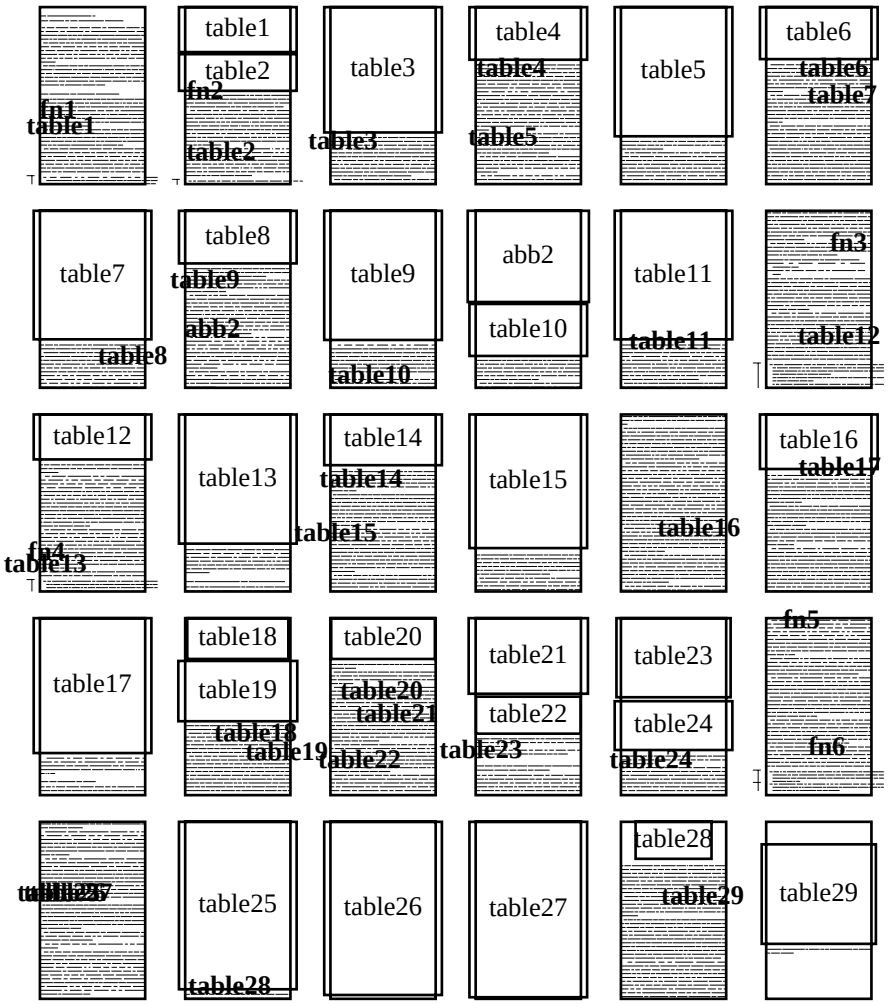

Figure 11: The first-fit pagination of *doc3*

To illustrate the difference our algorithm can make we depict symbolically the status of *doc3* before and after optimization. Figure 11 shows what a first-fit algorithm (as built into Microsoft Word) had been able to achieve. On the other hand, Figure 12 shows the *Turn.D*-90% optimal pagination of *doc3*. It needs one page more but its quality is much higher. One can observe that pages 4 and 5 (at the top right corner) are not 100 percent full but balanced in page height.

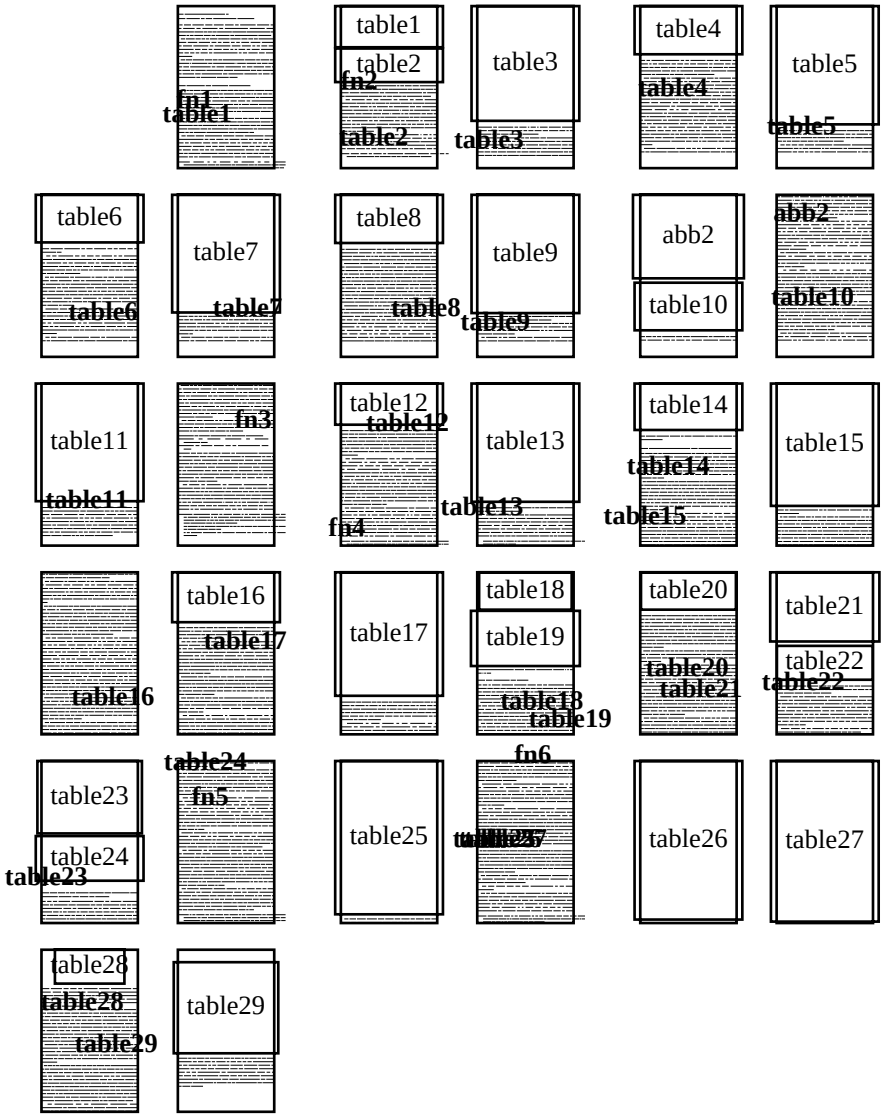

Figure 12: An optimal double-sided pagination of *doc3* with pages at least 90% full

This example also contains footnotes. Although not explicitly mentioned in this work, it is possible to extend the measure of quality and the pagination algorithm to cope with footnotes, too. Details about the extension may be found in Wohlfeil [1997].

5.2 Pagination Time

Needing an extra page is not the only price one has to pay to get better paginations. An optimizing algorithm needs more storage capacity and more running time to compute its results.

In Wohlfeil [1997] we have shown the techniques that were used to speed up the pagination program: (1) An efficient storage structure for the results of the subproblems. (2) A pruning heuristic to avoid computations that probably do not lead to the optimal solution. With these two techniques it was possible to compute a pagination in (at most) about one minute. In these experiments, a SUN Ultra I workstation with a 167 MHz processor and 256 MB of main memory has been used on the example documents. As compared to the many hours that authors today have to spent on manually improving the pagination of their documents, we think that our pagination algorithm is fast enough for daily use.

Although we did not paginate a whole book, our examples are still realistic. Namely, in practice there is no need to deal with a book as a whole because each chapter starts on a new page. Thus, the chapters may be paginated one by one, and then the resulting pages can be concatenated.

6 Conclusions and Further Work

In this work we have discussed the pagination problem of complex documents. From a theoretical point of view, we have proven why existing systems are bound to fail. Practically, we have presented a new measure of pagination quality, an optimizing pagination algorithm, and a prototype implementation that has shown very promising results.

An interesting question is if it will be technically possible to embed our algorithm into one of the major existing formatting systems; here we are widely open to suggestions and cooperation. Another approach could be to extend our prototype, XFORMATTER, into a fully-fledged formatting system.

Often the pagination of documents is not done by computer scientists but by people who have a background in arts. Therefore, the interface of a pagination program should be easy to use. This is all the more important as we do not expect that designers will always be satisfied with the very first pagination suggested by the system.

In such cases, the user should be provided with ways of interactively influence what the pagination algorithm does. This can be done by specifying constraints. It would be interesting to know if the two kinds of additional constraints currently implemented in XFORMATTER—under-full pages and ways to force two boxes on the same or on different spreads or pages— are sufficient. Here we hope to benefit from further experience.

References

ASHER, G. 1990. Type & Set: TEX as the Engine of a Friendly Publishing System. *Pages 91–100 of:* CLARK, M. (ed), *TEX: Applications, Uses, Methods.* Chichester, UK: Ellis Horwood Publishers. Proceedings of the TEX88 Conference.

BELLMAN, RICHARD. 1957. *Dynamic Programming.* Princeton, New Jersey: Princeton University Press.

BRÜGGEMANN-KLEIN, A., KLEIN, R., & WOHLFEIL, S. 1996. Pagination Reconsidered. *Electronic Publishing—Origination, Dissemination, and Design*, **8**(2&3), 139–152.

CHI82. 1982. *The Chicago Manual of Style.* The University of Chicago Press, Chicago.

CORMEN, T. H., LEISERSON, C. E., & RIVEST, R. L. 1990. *Introduction to Algorithms.* Cambridge, MA: MIT Press.

Dyson, Mary C., & Kipping, Gary J. 1997. The Legibility of Screen Formats: Are Three Columns Better Than One? *Computers & Graphics*, **21**(6), 703–712.

EFI95. 1995 (July). *Wissenschaftliche Information im elektronischen Zeitalter. Stand und Erfordernisse*. Bayerisches Staatsministerium für Unterricht, Kultus, Wissenschaft und Kunst, RB-05/95/14. Available on the WWW by the URL http://www11.informatik.tu-muenchen.de/EFI/. Bericht der Sachverständigenkommission zur elektronischen Fachinformation (EFI) an den Hochschulen in Bayern.

Fiat, Amos, & Woeginger, Gerhard (eds). 1996. *On-line Algorithms: The State of the Art*. Springer-Verlag.

Furuta, Richard, Scofield, Jeffrey, & Shaw, Alan. 1982. Document Formatting Systems: Survey, Concepts and Issues. *Pages 133–210 of:* Nievergelt, J., Coray, G., Nicoud, J.-D., & Shaw, A.C. (eds), *Document Preparation Systems*. Amsterdam: North Holland.

Garey, M. R., & Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY: W. H. Freeman.

Hansen, Wilfred J., & Haas, Christina. 1988. Reading and Writing with Computers: A Framework for Explaining Differences in Performance. *Communications of the ACM*, **31**(9), 1080–1089.

Kanders, Michael. 1996. *Schule und Bildung in der öffentlichen Meinung*. Beiträge zur Bildungsforschung und Schulentwicklung. Dortmund, Germany: IFS-Verlag.

Kernighan, B. W., & Wyk, Ch. J. Van. 1989. Page Makeup by Postprocessing Text Formatter Output. *Computing Systems*, **2**(2), 103–132.

Klein, R. 1997. *Algorithmische Geometrie*. Bonn: Addison-Wesley.

Knuth, Donald E. 1986. *The TEXbook*. Computer & Typesetting, vol. A. Reading, MA: Addison Wesley Publishing Company.

Knuth, Donald E., & Plass, Michael F. 1981. Breaking Paragraphs into Lines. *Software—Practice and Experience*, **11**, 1119–1184.

Lamport, L. 1986. *LATEX: A Document Preparation System, User's Guide & Reference Manual*. Reading, MA: Addison-Wesley Publishing Company.

Pagel, Bernd-Uwe, & Six, Hans-Werner. 1994. *Software Engineering; Band 1: Die Phasen der Softwareentwicklung*. Reading, Massachusetts: Addison-Wesley.

Piolat, Annie, Roussey, Jean-Yves, & Thunin, Olivier. 1997. Effects of screen presentation on text reading and revising. *Int. J. Human-Computer Studies*, **47**, 565–589.

Plass, M. F. 1981. *Optimal Pagination Techniques for Automatic Typesetting Systems*. Technical Report STAN-CS-81-870. Department of Computer Science, Stanford University.

Plass, M. F., & Knuth, D. E. 1982. Choosing Better Line Breaks. *Pages 221–242 of:* Nievergelt, J., Coray, G., Nicoud, J.-D., & Shaw, A. C. (eds), *Document Preparation Systems*. Amsterdam: North Holland.

Rubinstein, R. 1988. *Digital Typography: An Introduction to Type and Composition for Computer System Design*. Reading, MA: Addison-Wesley Publishing Company.

Sleator, D. D., & Tarjan, R. E. 1985. Amortized efficiency of list update and paging rules. *Commun. ACM*, **28**, 202–208.

Williamson, H. 1983. *Methods of Book Design*. New Haven: Yale University Press.

Wohlfeil, Stefan. 1997 (Dec). *On the Pagination of Complex, Book-Like Documents*. Ph.D. thesis, Fernuniversität Hagen, Germany.