Optimal Line Breaking in Music

Wael A. Hegazy, John S. Gourlay

Department of Computer and Information Science
The Ohio State University

This work was partially supported by the National Science Foundation under grant number IST-8514308.

Abstract

For a variety of reasons music line breaking, or "casting off", as it is called by musicians, is not susceptible to simple treatment in automatic music formatting software. The powerful line breaking method used by Knuth in the text formatting program TEX can almost do the job for music, but it falls short in being unable to handle the complex motions of notes when lines of music are stretched or shrunk. This paper describes a generalization of the line-breaking model used in TEX in which springs (analogous to TEX's glue) penetrate boxes and connect to interior points. A line-breaking algorithm based on this model is described, and implementation details that make it computationally feasible are discussed.

INTRODUCTION TO THE PROBLEM

Among the many "exotic" document formatting problems currently receiving attention, music formatting is one of the most challenging. For some time, the MusiCopy project at The Ohio State University has been searching for algorithms to automate many of the kinds of formatting decisions made routinely by human music copyists in preparing high-quality musical scores [3]. One of the recent investigations has been into the problem of line breaking, or "casting off" as it is called by musicians.

It is tempting to dismiss the music line-breaking problem as a simple variation of text line breaking, treating measures like words, and barlines like spaces between words. Careful consideration of the notational conventions of music, however, reveals several reasons why a simple approach will not work.

A fairly evident feature of printed music is that the last line of a piece or movement is always full. Unlike a paragraph of text, a piece of music conventionally cannot end with a short line, and music copyists must be willing to revise early line-breaking decisions in the likely case that their initial choice of breaks does not lead to a full last line. Automated line breaking for music must exhibit the same flexibility in looking ahead or backtracking, or it will be forced to stretch or shrink the last line by intolerable amounts. It is also considered desireable that long works of music fill the whole last page, causing copyists sometimes to set an entire piece slightly tighter or looser than they would otherwise to force the piece to fit in a number of lines that is a multiple of the number of lines per page.

A related issue in music is that certain page breaks and sometimes line breaks can be very troublesome for performers, who must be able to deliver a smooth performance from music they have not rehearsed thoroughly. We cannot expect automated systems to detect these breaks, but it must be possible for users of music printing software to be able to encourage or discourage certain line and page breaks while the software maintains uniform spacing throughout the piece.

Another complexity of music is that certain bits of text, for example clefs and key signatures, are repeated at the beginning of every line. It is possible for these features of the music to change from time to time during a piece, not only in appearance but in size, and so they must be introduced and accounted for by a line-breaking algorithm rather than treated as a constant background on which the rest of the music is printed.

A last and very important feature of music relevant to line breaking is the complex motion of notes with respect to one another as the line they are in is stretched or shrunk. In text we desire uniform spaces between words, regardless of the widths of the words themselves. In music, on the other hand, the important distances are from the left edge of one note to the left edge of the next, leaving variable amounts of white space depending on the physical widths of the notes. These edge-to-edge distances depend on the durations of the notes. This in itself would not be difficult to handle, but notes must not be allowed to overlap, even in tight spacings where the edge-to-edge distance between a pair of notes could otherwise become too small. This behavior will be discussed in more detail later, but the important consequence is that the position of a note in a line is not usually a simple linear function of the amount by which the line is stretched or shrunk, and a line-breaking algorithm for music must take this into account in fitting measures into lines.

The difficulties of casting off music have not been overlooked by others, but there appears to be no published algorithm that attempts to deal with them. Gomberg [2] and Byrd [1], both of whom have described algorithms for automatic music formatting, discuss the problems but simply defer any useful line-breaking decisions to the users of their software. Byrd's software will do "first-fit" line breaking (in which each line is filled as full as possible and never reconsidered), but he suggests that his users will usually want to choose their own breaks. Ross [7], as well as others writing for musicians and copyists, offers little guidance other than to avoid certain musically bad breaks, and to keep spacing as uniform as possible.

The goal of this paper is to describe an approach to music line breaking that can successfully deal with all of the above problems. The algorithm is a generalization of the one devised by Knuth for the text formatter TEX, and so the next section contains a brief summary of Knuth's approach. The third section introduces the necessary generalizations to Knuth's model, and the last two sections describe the line-breaking algorithm itself and discuss its efficiency.

SUMMARY OF THE TEX LINE-BREAKING METHOD

One of several contributions to text formatting made by Knuth in his program TEX is a fast line-breaking algorithm that takes a global view of the paragraph, rather than proceeding line by line [5]. At the heart of the algorithm is a "cost" function that, given a particular choice of line breaks for a paragraph, yields a numerical cost in units called *demerits*. The cost function is designed to take into account several features of typographically good paragraphs, producing low costs for good paragraphs and higher costs for bad paragraphs. The job of the line-breaking algorithm, then, is to find a series of line breaks that minimizes the cost of the paragraph.

For the purpose of Knuth's algorithm, a paragraph before line breaking is a sequence of *boxes* with *springs* between them. (Knuth calls the springs "glue," but "spring" is a better metaphor for the generalization to music.) The boxes correspond to words and have fixed widths. The springs correspond to the spaces between words, and they have ideal widths as well as separate values giving their stretchability and shrinkability.

Keeping the physical analogy in mind, we can see that a sequence of boxes and springs will have a natural width equal to the sum of the widths of the boxes and the ideal widths of the springs between them. By applying a force to stretch or shrink the sequence, we can change the width of the sequence, but the more we deviate from the natural width of the sequence, the greater the force will have to be.

Whatever force we apply, all of the boxes in the sequence will take well-defined positions so that the forces on all of the springs are identical. This fact makes the relation between the force and the change in length a very easy one. For example, if Y is the sum of the stretchabilities of all the springs in the sequence, then the *fitting force* required to increase the length of the sequence by an amount s is what Knuth calls the "adjustment ratio" $r = s/Y$.

To compute the cost of a set of line breaks in a paragraph, basically TEX computes the force required to make each line the right length, and then takes the sum of the sixth powers of these forces. In minimizing this cost function, the sixth powers place an extremely high penalty on variations in force throughout the paragraph, thus satisfying the goal of uniformity of spacing. Actually, the cost function takes other typographical qualities into account as well in the form of *penalties* attached to each line break, either by the author or automatically, for example when TEX detects hyphenations on two consecutive lines. Still overlooking some technicalities, the cost function more precisely adds the penalties to the cubes of the forces, and the results are then squared and summed.

Of course, it would be prohibitively expensive to try to enumerate all possible sets of line breaks for a paragraph in order to find the one with the minimum cost. This can be avoided in the case of Knuth's cost function because if we fix a break point in a paragraph, the optimality of a set of break points prior to this point is unaffected by choices of break points beyond this point. Thus we can proceed through the paragraph one possible break point at a time, remembering only the optimal way of getting to each prior legal break point, and finding the optimal way of getting to the new break point using only this saved information. An additional economy is obtained in TEX by putting a limit on the amount of force that can be applied to any line. This limit removes the majority of possible lines from consideration because they are stretched or shrunk by intolerable amounts. Thus, when it is considering a particular break point, TEX's algorithm keeps only a short *active list* of prior break points and the optimal ways to reach them, the active list being limited to only those points starting lines reaching to the current point with tolerable amounts of stretch or shrink.

A MUSIC LINE-BREAKING MODEL

It turns out that Knuth's line-breaking model can handle all of the problematical features of music except the last. If there is no spring at the end of a paragraph, the algorithm will automatically find an optimal setting with a full last line. If there is a particular word space at which an author would like to discourage a line break, he or she need only specify an appropriate penalty value at that point. Although it is not obvious from the foregoing, TEX can also force a paragraph into a specific number of lines, satisfying the need to fill a last page of music, or alternatively the whole algorithm can be applied a second time to find optimal page breaks as was done by Plass [6]. Finally, with a simple and powerful hyphenation feature called a *discretionary break*, a line break can be designed to modify the texts at the end of the previous line and the beginning of the new one.

This leaves only the problem of the relative motions of notes during stretching and shrinking. As we have already said, these do not obey the simple linear rule followed by text. In Figure 1 we see three versions of the same measure, the first one severely shrunk, the second at its natural width, and the third severely stretched. If we look first at the pair of eighth notes (the two notes connected by a horizontal bar), we see that their spacing changes smoothly with the line length. If notes never had large printed widths, this is what should happen in all cases, but if we look at the last pair of notes we see that the space between these notes is the same in both the shrunk and natural versions of the measure. If we imagine an animation of the measure shrinking from its maximum to its minimum size, the space between these notes shrinks smoothly until about the time the measure reaches its natural width. At this point the pair of flats in front of the last note abruptly stops any shrinkage of the space. An extreme case of this occurs between the first two notes, whose spacing never changes at any line length. The flats belonging to the second note are so wide that they always require more space than would be appropriate for this pair of notes in the absence of the flats. (The ideal spacings of notes depend on their durations in a logarithmic fashion described elsewhere [4], so we will not concern ourselves here with how the numbers are arrived at.)

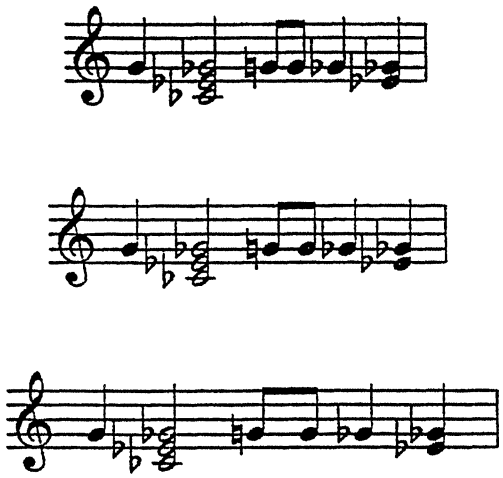

Figure 1.

Evidently, then, the physical analogy to boxes and springs still applies, but the attachment points of the springs must be changed. Notes and any associated accidentals (flats, sharps, or natural signs) occupy boxes, just as words do, but the springs connecting them do not connect to the outsides of the boxes, but penetrate the boxes and connect to common points at the left edges of the noteheads. While springs can penetrate boxes, boxes cannot penetrate each other, and when they come into contact they prevent springs from shrinking as far as they would otherwise.

Every spring still has values for ideal width, stretchability, and shrinkablility, but when we consider trying to stretch or shrink a line of this sort by a given amount, the computation of the required force

becomes complex, because as the line gets shorter, fewer and fewer springs are available to absorb the reduction in width. The computation can be done, however, and the cost function derived from this fitting force is our basis for optimal line breaking in music.

OVERVIEW OF THE ALGORITHM

Introduction

Under this model, the input to the line-breaking algorithm consists of a sequence of *items* describing the behavior of the springs connecting the notes. Each item has four parts, called w, y, z, and b, where w is the ideal width of the spring, y is its stretchability, z is its shrinkability, and b is the width at which the two boxes it connects collide, what we call the ***blocking width*** of the spring. If $b > w$ we say that the spring is *prestretched*. Interspersed among these items are penalties, indicating the locations of ***legal break points***, and which in practice will occur just prior to barlines.

As we said, the problem of line breaking is one of finding a sequence of legal breakpoints which results in a sequence of lines whose total demerits is minimum. But since a piece of music that has m measures would have 2^m possible sequences of breakpoints, it is evident that the approach that generates all the possible sequences of breakpoints should be discarded.

As in text, however, most of the possible sequences of breakpoints are absurd, in the sense that they would result in lines that either are packed with so many measures that they can never fit into the desired line width, or have very few measures that they are ridiculously loose. Therefore, the number of sequences of breakpoints that the line-breaking algorithm need to consider can be brought down significantly by defining a permissible range of the force needed to fit a sequence of measures into a desired line width (or equivalently, by defining a certain tolerance for line badness), and dropping out of consideration any sequence that would result in a line whose fitting force is outside the permissible range. (We will denote such a range by $[f_{\min}, f_{\max}]$, where $f_{\min}$ has a negative value and $f_{\max}$ has a positive value, following the convention that a stretching force is positive and a shrinking force is negative.) However, the number of "reasonable" sequences of breakpoints can still be out of the scope of practicality for yet reasonable values of $f_{\min}$ and $f_{\max}$.

Luckily, dynamic programming techniques can be applied to the problem so that the "reasonable" sequences of breakpoints can be compared dynamically while they are being constructed. That is, when a potential breakpoint belongs to different such sequences, the algorithm needs to keep track only of the best subsequence that leads to this breakpoint; the other subsequences that lead to it can be dropped out of consideration completely. Therefore, at any given time during the process of constructing and comparing the sequences of breakpoints, the number of subsequences that are kept track of usually is very small. It will be shown, shortly, that this number is bounded by the maximum number of measures per line.

Although the dynamic programming techniques, along with defining a certain tolerance for line badness, might be sufficient to bring the optimum line-breaking algorithm to practical efficiency for text, it is still not as sufficient in the case of breaking musical scores into lines. This is, mainly, due to the fact that computing the badness of a line in the case of musical scores is not as simple as it is in the case of text. The rest of this section describes the outline of the music line-breaking algorithm, which does not differ in any significant way from the outline of the text line-breaking algorithm used in TEX. A subsequent section of this report presents ways for gaining more efficiency for the music line-breaking algorithm (henceforth, will be called just "the algorithm".)

The Main Idea

The main idea of the algorithm is to examine each legal breakpoint to see if the part of the piece from the beginning to this point can be broken into lines each of which has a fitting force within the permissible range (such legal breakpoint is called "feasible breakpoint"). Whenever a feasible breakpoint is found, the algorithm remembers, from among all the sequences of feasible breakpoints that lead to the current one, the sequence that results in the fewest total demerits from the beginning of the piece to the current feasible breakpoint. Note that in order to remember such a sequence, it suffices to remember, for each feasible breakpoint, only the feasible preakpoint that immediately precedes it in the sequence.

To be able to check whether a legal breakpoint is feasible or not, the algorithm maintains a list of the feasible breakpoints that might mark the beginning of a potential line whose end will be at a future breakpoint. This list is called the list of active breakpoints, or just the "active list." Initially, the active list contains only the breakpoint that marks the beginning of the first line. A legal breakpoint b is examined for being a feasible breakpoint by checking if there is any active breakpoint a such that the fitting force of the potential line from a to b is within the permissible range. If so, b is a feasible breakpoint and it is appended to the active list, along with an indication of the active breakpoint a that minimizes the total demerits from the beginning of the piece to breakpoint b. Otherwise, the potential line from a to b is either too short or too long to fit into the desired line width using a fitting force within the permissible range. If the potential line is too short, the algorithm proceeds to the next legal breakpoint since any line ending at breakpoint b and starting at an active breakpoint after a would certainly be too short too. On the other hand, if the potential line from a to b is too long, a is removed from the active list since no future breakpoint would ever mark the end of a line that breakpoint b marks its beginning. Recalling that legal breakpoints can be only between measures, the size of the active list, therefore, is bounded by the maximum number of measures in a line.

As an example, Figure 2 shows a piece of music with little numbered marks indicating the legal breakpoints. Figure 3 shows a partial tracing of the algorithm, when applied to the piece of Figure 2, indicating the potential lines examined and the changes in the active list along the way. Figure 4 shows a graph that represents the computation of the algorithm on the piece of Figure 2. The nodes of the graph represent the feasible breakpoints. The edges, both the arrowed and the non arrowed ones, represent feasible lines between pairs of feasible breakpoints. There is exactly one arrowed edge going out of each node b; it points to the node representing the feasible breakpoint that immediately precedes b on the best sequence of feasible breakpoints from the beginning of the piece to b. The number next to each edge represents the demerits of the line corresponding to the edge.

Assuming that the whole piece can be broken into lines each of which has a fitting force within the permissible range (which is the normal case), then the breakpoint that marks the end of the piece is a feasible breakpoint. Therefore, like any other feasible breakpoint, the algorithm appends this last feasible breakpoint to the active list, and remembers its preceding feasible breakpoint in the best breakpoint sequence that leads to it. Hence, to determine the optimum breakpoint sequence for the piece, the algorithm just needs to follow the backward links starting at the last breakpoint in the active list (the back arrows in the graph of Figure 4.) It is worth mentioning here that the line breaking decided by the algorithm for the piece in Figure 2 coinsides with the line breaking decided by a professional musician for the same piece. It took a few hours to cast off the piece manually, whereas an implementation of the algorithm on a personal computer did the job in a few seconds.

A simplified general outline of the algorith has the following form:

```
        :
⟨ for every legal breakpoint b ⟩ do
  begin
  ⟨ Initialize the feasible breakpoints at b to the empty set ⟩
  ⟨ for every active breakpoint a ⟩ do
    begin
    ⟨ Compute the fitting force f for the potential line from a to b ⟩
    if f < f_min then ⟨ remove a from the active list ⟩
    elseif f > f_max then ⟨ exit inner loop ⟩
    else ⟨ record a feasible break from a to b ⟩
    end
```

```
⟨ if there is a feasible break at b ⟩ then
    ⟨ append the best such break to the active list ⟩
  end
⟨ Use the last feasible break in the active list to
        trace back the optimum breakpoint sequence ⟩
        ⋮
```

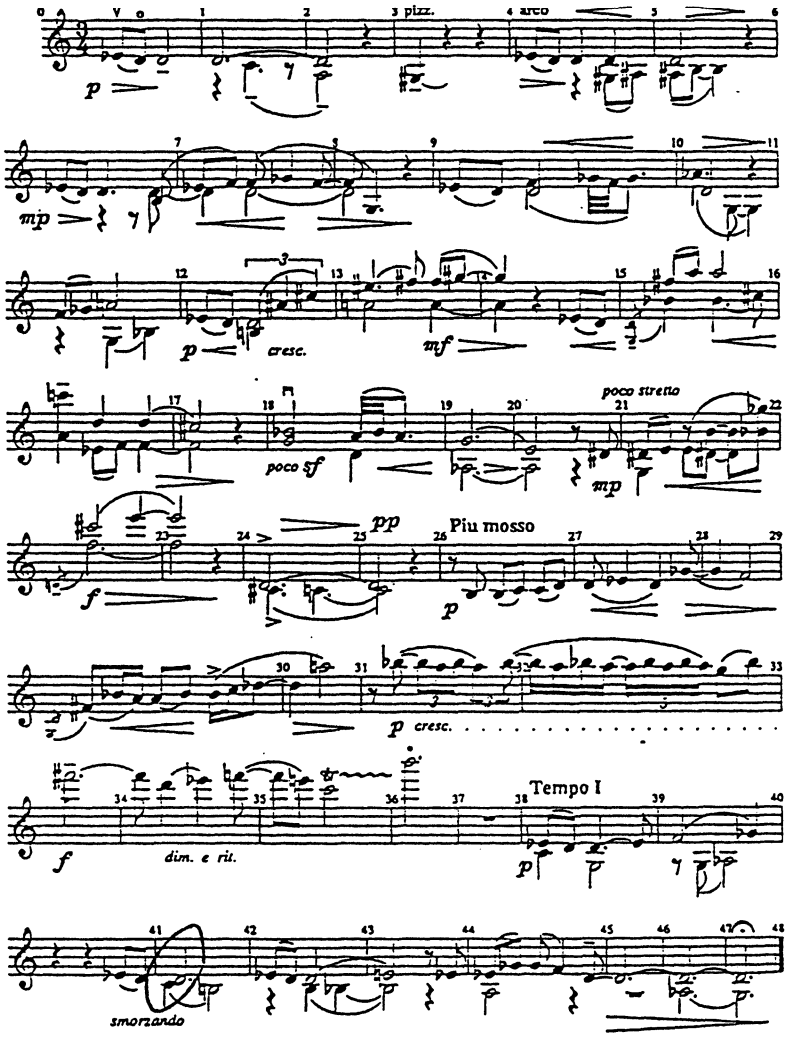

Figure 2.

Since legal breakpoints occur only between measures, and since the size of the active list is bounded by the maximum number of measures per line, the number of times the body of the inner loop of the algorithm is executed is bounded by $mpp \times mpl_{\max}$, where mpp is the number of measures in the piece and $mpl_{\max}$ is the maximum number of measures per line. The main operation inside the inner loop is that of computing the fitting force for the measures between a feasible breakpoint a and a legal breakpoint b. This operation

Potential line		Status	Set of a's that start a feasible line ending at b	Best a	Changes in the active list
Legal b.p. b	Active b.p. a				
					[0]
1	0	too short			
2	0	too short			
3	0	too short			
4	0	too short			
5	0	too short			
6	0	feasible	{0}	0	[0, 6]
7	0	feasible	{0}		
7	6	too short	{0}	0	[0, 6, 7]
8	0	too long			[6, 7]
8	6	too short			
9	6	too short			
10	6	feasible	{6}		
10	7	too short	{6}	6	[6, 7, 10]
			⋮		
27	22	too short	{21}	21	[21, 22, 23, 24, 25, 27]
28	21	feasible	{21}		
28	22	feasible	{21, 22}		
28	23	too short	{21, 22}	22	[21, 22, 23, 24, 25, 27, 28]
29	21	too long			[22, 23, 24, 25, 27, 28]
29	22	feasible	{22}		
29	23	feasible	{22, 23}		
29	24	too short	{22, 23}	22	[22, 23, 24, 25, 27, 28, 29]
30	22	too long			[23, 24, 25, 27, 28, 29]
			⋮		

Figure 3. Partial tracing of the algorithm when applied to the piece of Figure 2.

the fitting force for the measures between a feasible breakpoint a and a legal breakpoint b. This operation should be made as efficient as possible since the efficiency of the whole algorithm depends, mainly, on the efficiency of this operation. The rest of this section summarizes the characterization of the fitting force needed to fit a given group of measures (or items) into a desired length.

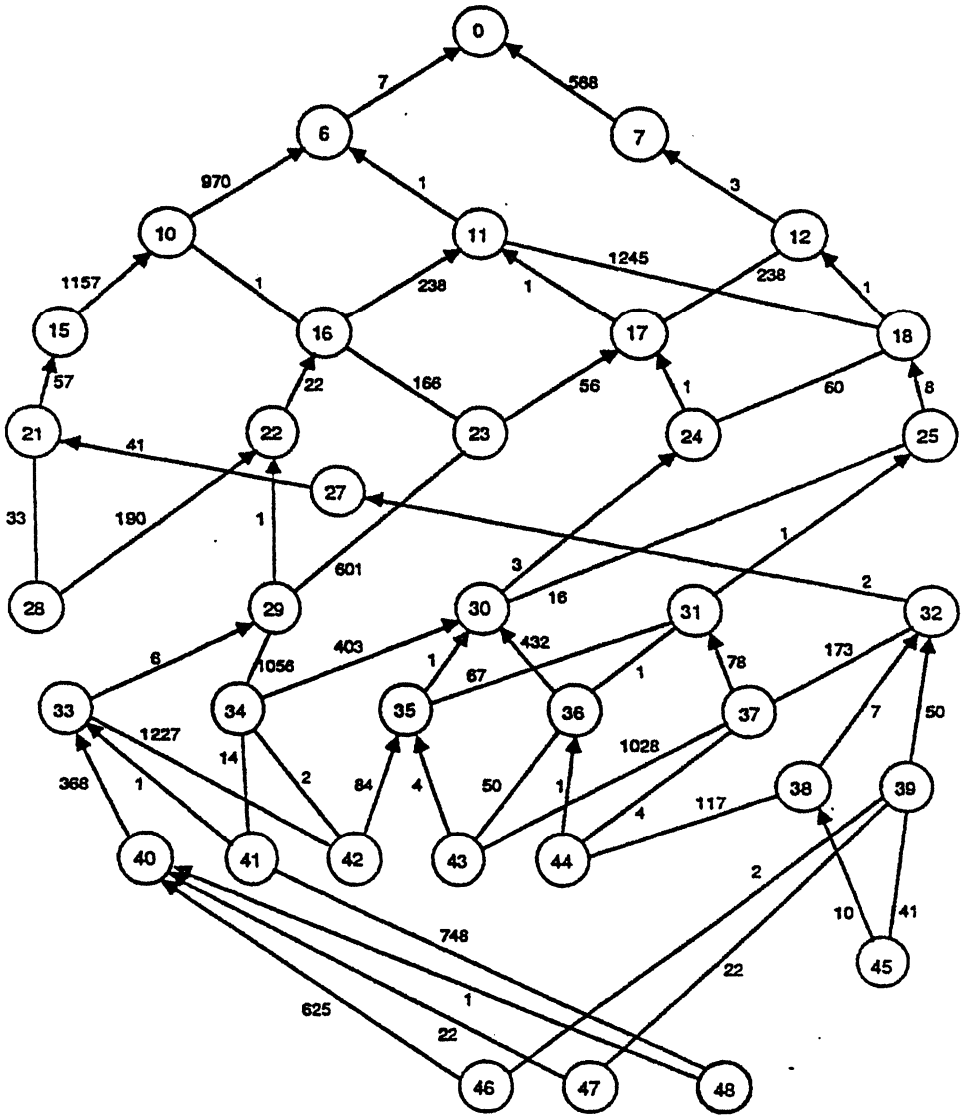

Figure 4. This graph represents the computation of the algorithm on the piece of Figure 2. The demerits of the feasible lines are rounded off for simplicity. An arrowed edge from node b to node a means that a immediately precedes b in the optimum sequence of breakpoints from the beginning of the piece to b. The optimum breakpoint sequence of the whole piece is determined by following the arrows, starting from the last node.

Finding The Fitting Force

The problem of finding the fitting force can be formulated as follows: given a sequence of items, I, and an amount of stretch or shrinkage, s (to fit the potential line into the desired line width), what is the necessary fitting force, f, that is needed to stretch or shrink the whole group of items I by s? We will adopt the convention that both s and f are positive for stretch and are negative for shrinkage. And since f does not depend on the order of items in I, we will refer to I as a set rather than a sequence. Each item in I is represented as a 4-tuple for the w, y, z, and b values of the item. The idea behind finding the fitting

force can probably be understood best in terms of a function that maps an amount of stretch or shrinkage, for a set I of items, into the required fitting force. Such a function will be denoted by $\phi_I(s)$. The function $\phi_I(s)$ is a piece-wise linear function whose line segments depend on I. for the purpose of defining $\phi_I(s)$ for an arbitrary set of items I, the set I is partitioned into three disjoint sets, I_{PR}, I_{SNP} and I_{NSNP} as follows:

I_{PR}: Contains the prestretched items in I; i.e., those items for which $y > 0$ and $b > w$. The elements of I_{PR} are denoted by $(w_1, y_1, z_1, b_1), \ldots, (w_m, y_m, z_m, b_m)$. Without loss of generality, it can be assumed that:

$$\frac{b_1 - w_1}{y_1} \leq \frac{b_2 - w_2}{y_2} \leq \cdots \leq \frac{b_m - w_m}{y_m}$$

I_{SNP}: Contains those non-prestretched items in I that can shrink; i.e., those items for which $b < w$ and $z > 0$. The elements of I_{SNP} are denoted by $(w'_1, y'_1, z'_1, b'_1), \ldots, (w'_n, y'_n, z'_n, b'_n)$. Without loss of generality, it can be assumed that:

$$\frac{w'_1 - b'_1}{z'_1} \leq \frac{w'_2 - b'_2}{z'_2} \leq \cdots \leq \frac{w'_n - b'_n}{z'_n}$$

I_{NSNP}: Contains those non-prestretched items in I that cannot shrink; i.e., those items for which $b = w$, or $b < w$ and $z = 0$. The elements of I_{NSNP} are denoted by $(w''_1, y''_1, z''_1, b''_1), \ldots, (w''_p, y''_p, z''_p, b''_p)$.

Using the above notation, the function $\phi_I(s)$, for an arbitrary I, is represented graphically in Figure 5. The function $\phi_I(s)$ will be called the characteristic function of I, or the characteristic function of the potential line whose items make up the set I. According to the specifications of $\phi_I(s)$, as shown in Figure 5, it is apparent that finding the fitting force for some I and s inevitably involves two main steps. In the first step, the elements of the set I are scanned so that I can be partitioned into I_{PR}, I_{SNP} and I_{NSNP}. The sums $\sum y + \sum y''$ or $\sum z'$ might be computed while the elements of I are being scanned. The second step involves constructing as many line segments, among those which represent $\phi_I(s)$, as necessary to reach the segment that corresponds to the given value of s.

IMPROVING THE ALGORITHM'S EFFICIENCY

According to the outline of the algorithm given in the preceding section, the fitting force is computed for every potential line. This would take too much time unnecessarily. In the first place, a large portion of the potential lines are not feasible lines (i.e., they need fitting forces outside any "reasonable" permissible range), and the time needed to compute the fitting forces for these lines can be saved if an efficient way is used to filter out the infeasible lines. Furthermore, computing the fitting forces for the feasible lines can be made more efficient, on the average, by taking advantage of the fact that many of these feasible lines need fitting forces within the first positive or the first negative line segment of $\phi_I(s)$. Computing the fitting force that is within the first line segment of $\phi_I(s)$ can be made much more efficient if it is known, beforehand, that it is indeed within the first segment. More efficiency, therefore, can be obtained if a simple test can be used to determine whether the fitting force for a given line is within the first line segment of the line's characteristic function.

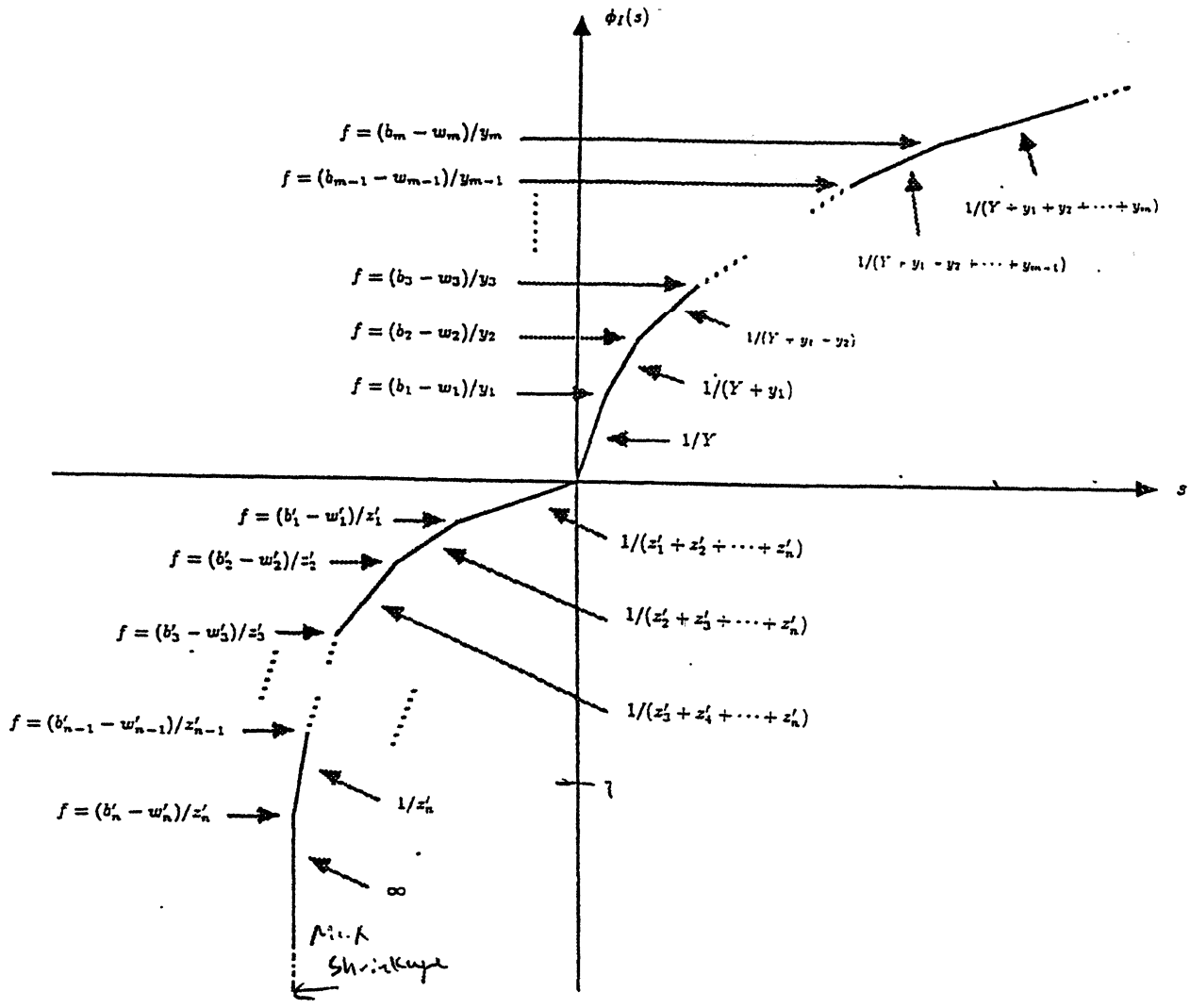

Figure 5. This graph represents $\phi_I(s)$ for an arbitrary I. The expressions next to the arrows pointing at the boundaries between line segments represent the values of the force at these points. The expressions next to the arrows pointing to the line segments themselves represent the slopes of the segments, with $Y = \sum_{i=1}^{n} y'_i + \sum_{i=1}^{p} y''_i$.

Filtering Infeasible Potential Lines

A potential line is feasible only if its fitting force is within a given permissible range, $[f_{\min}, f_{\max}]$. It might seem, therefore that the fitting force of a potential line must be computed in order to determine whether it is a feasible line or not. Fortunately, this is not true. For example, depending on whether the potential line is to be stretched or shrunk, one can find $s_{\max}$ (or $s_{\min}$) corresponding to stretching (or shrinking) with force $f_{\max}$ (or $f_{\min}$). And since the function $\phi_I(s)$ can be shown to be always monotonic, it can be concluded that the fitting force needed for a potential line is within the permissible range if and only if $s_{\min} \leq s \leq s_{\max}$, where s is the amount of stretch (or shrinkage) needed to fit the potential line into the desired line length.

This way of filtering out the infeasible potential lines, however, does not seem to be efficient enough.

The purpose of such filtering is to save the time of computing the fitting force in the cases where its exact amount is not going to be used. It, therefore, makes no sense to do this filtering in a way that might take more time than the time needed for computing the exact fiting force. This can be the case here due to the fact that many of the infeasible potential lines need fitting forces that are within the first line segment of their characteristic functions, in which case the fitting force can be computed most efficiently. Therefore, a more efficient way for filtering out the infeasible potential lines is sought.

A better approach towards filtering out the infeasible potential lines would be to approximate $\phi_I(s)$ in such a way that makes computing the 'approximate' fitting force efficient enough for the filtering purpose. Let $\phi_I^A(s)$ be such an approximation of $\phi_I(s)$. The approximation must be such that no potential line that is actually feasible would ever be considered infeasible by the approximation. In other words, for every given value of s, it must be true that $|\phi_I^A(s)| \leq |\phi_I(s)|$. An appropriate function $\phi_I^A(s)$ would approximate $\phi_I(s)$ by two line segments as follows:

$$\phi_I^A(s) = \begin{cases} s/(\sum_{i=1}^{n} y_i' + \sum_{i=1}^{m} y_i + \sum_{i=1}^{p} y_i''), & \text{if } s \geq 0; \\ s/\sum_{i=1}^{n} z_i', & \text{otherwise.} \end{cases}$$

As shown in Figure 6, $\phi_I^A(s)$ satisfies the condition that $|\phi_I^A(s)| \leq |\phi_I(s)|$ for any given value of s. Besides, $\phi_I^A(s)$ is such a simple function that filtering out infeasible potential lines using it is quite efficient and independent of the number of items in the line, unlike the case when $\phi_I(s)$ is used. Note that the sums $\sum y'$, $\sum y''$, $\sum y$, and $\sum z'$ need not be computed, over and over, for each potential line. Instead, the sums are computed from the beginning of the piece to the current place, and two such sums are subtracted to obtain the sum for what lies between them.

Filtering infeasible potential lines using $\phi_I^A(s)$ is an imperfect filtering, in the sense that some infeasible lines are not filtered out. In particular, for any value of s where $\phi_I^A(s) < f_{\max} \leq \phi_I(s)$ (in case of stretch), or $|\phi_I^A(s)| < |f_{\min}| \leq |\phi_I(s)|$— (in case of shrinkage), a line is considered feasible when $\phi_I^A(s)$ is used for computing the fitting force, although the line, in fact, is infeasible. This imperfection in filtering infeasible lines does not represent a problem; an infeasible line that is not discarded sooner using $\phi_I^A(s)$ will be discarded later, after its exact fitting force is computed using $\phi_I(s)$. Nevertheless, the probability of detecting infeasible lines using $\phi_I^A(s)$ is high enough to make the overall performance of the algorithm with the imperfect filtering, using $\phi_I^A(s)$, better than it with the perfect filtering, using $\phi_I(s)$.

Checking For The First-segment Cases

Computing the fitting force that is within the first line segment of $\phi_I(s)$ can be made most efficient if it is known, beforehand, that it is indeed within the first segment. In this case the fitting force f is as follows:

$$f = \begin{cases} 0, & \text{if } s = 0; \\ s/\sum_{i=1}^{n} y_i', & \text{if } s > 0; \\ s/\sum_{i=1}^{n} z_i', & \text{otherwise.} \end{cases}$$

And since $\sum_{i=1}^{n} y_i'$ and $\sum_{i=1}^{n} z_i'$ are computed anyway for each potential line, for the purpose of filtering out the infeasible potential lines, computing the fitting force that is known to be within the first segment of the line's characteristic function boils down to a single division operation.

Referring to Figure 5, the fitting force f is within the first segment if and only if $f \leq \min_{1 \leq i \leq m}(b_i - w_i)/y_i$, in case of stretch, or $|f| \leq \min_{1 \leq i \leq n}(w_i - b_i)/z_i$, in case of shrinkage. Given an amount s of stretch (or shrinkage), it can be determined whether the required fitting force f is within the first segment or not as follows:

$$f \text{ is within the first segment} \iff \begin{cases} s/\sum_{i=1}^{n} y_i' \leq \min_{1 \leq i \leq m}(b_i - w_i)/y_i, & \text{if } s \geq 0; \\ s/\sum_{i=1}^{n} z_i' \leq \min_{1 \leq i \leq n}(w_i' - b_i')/z_i', & \text{otherwise.} \end{cases}$$

The minima $\min_{1 \leq i \leq m}(b_i - w_i)/y_i$ and $\min_{1 \leq i \leq n}(w_i' - b_i')/z_i'$ could be computed, for every feasible potential line, by scanning through all the items in the line. However, experience has shown that it is more efficient

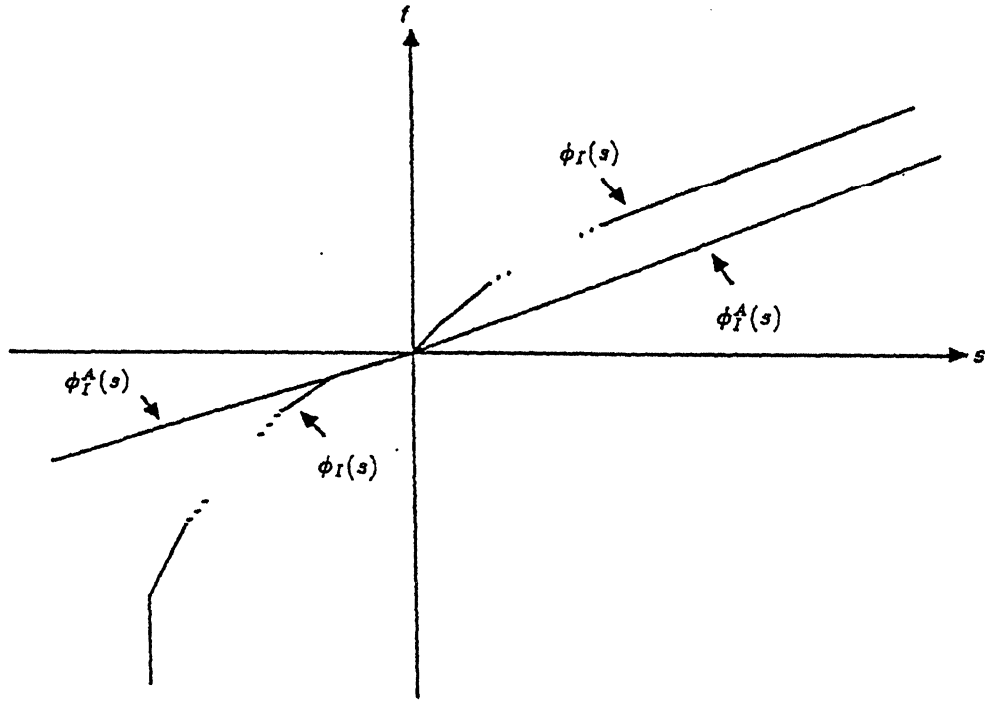

Figure 6. This graph depicts both $\phi_I(s)$ and $\phi_I^A(s)$. In the case of stretch ($s > 0$), $\phi_I^A(s)$ is a straight line that passes through the origin, and whose slope is equal to the slope of the last line segment of $\phi_I(s)$. In the case of shrinkage ($s < 0$), $\phi_I^A(s)$ is a straight line that passes through the origin, and whose slope is equal to the slope of the first line segment of $\phi_I(s)$. Using $\phi_I^A(s)$, the magnitude of the force corresponding to a given value of s is never overestimated.

to precompute these minima for each measure in the piece and then find the minima for each potential line by scanning through the measures of that potential line.

Checking every feasible potential line to see whether or not its fitting force is within the first segment of the line's characteristic function would save time only if the fitting force turns out to be within the first segment. Therefore, the overall saving in computation time depends on the probability that the needed fitting force is within the first segment. Such probability tends to get smaller as f_{max} and $|f_{min}|$ increase. For high values of f_{max} and $|f_{min}|$, it is possible that checking for the first-segment cases can have a negative effect in terms of computation time. However, a limited experience with using the music line-breaking algorithm has shown that for $f_{min} \geq -1$ and $f_{max} \leq 1$, the algorithm's performance is improved by checking for the first-segment cases.

BIBLIOGRAPHY

[1] Byrd, Donald A., *Music Notation by Computer*, Ph. D. Thesis, Indiana University, 1984.

[2] Gomberg, David A., *A Computer-Oriented System for Music Printing*, Ph. D. Thesis, Washington University, 1975.

[3] Gourlay, John S., "Computer Formatting of Music," *PROTEXT III Proceedings of the Third International Conference on Text Processing Systems*, J. J. H. Miller, ed., Boole Press, 1986.

[4] Gourlay, John S., "Spacing a Line of Music," To appear in *PROTEXT IV Proceedings of the Fourth International Conference on Text Processing Systems*, J. J. H. Miller, ed., Boole Press, 1987.

[5] Knuth, Donald E. and Plass, Michael F., "Breaking Paragraphs into Lines," *Software—Practice and Experience*, vol. 11, 1981, pp. 1119–1184.

[6] Plass, Michael F., *Optimal Pagination Techniques for Automatic Typesetting Systems*, Ph. D. Thesis, Stanford University, 1981.

[7] Ross, T., *The Art of Music Engraving and Processing*, Hansen Books, 1970.

ACKNOWLEDGEMENT

We would like to thank Dean K. Roush for his permission to use his composition "Lacrimosa" as an example in this paper, and for the contribution of his expertise in music notation.