Spacing a Line of Music

John S. Gourlay

OSU-CISRC-10/87- TR35

SPACING A LINE OF MUSIC

John S. Gourlay
Department of Computer and Information Science
The Ohio State University

October 8, 1987

SPACING A LINE OF MUSIC

John S. Gourlay

Abstract—This paper describes a very successful algorithm for choosing the spacing of notes in printed music given their durations and printed sizes. The algorithm is based on the two observations that the ideal spacing of notes is a logarithmic function of their durations, and that in the presence of several voices, the note with the shortest duration defines the space used for all simultaneous notes. The algorithm improves on previous ones in handling triplets and other complex rhythms simply and without special cases. It also allows for stretching and shrinking of lines of music for optimal line breaking and accomodating wide notes, maintaining the logarithmic spacing at all times. The results of the algorithm have been compared with good manual spacing of music, and the two are indistinguishable in the vast majority of cases.

SPACING A LINE OF MUSIC*

John S. Gourlay

Abstract—This paper describes a very successful algorithm for choosing the spacing of notes in printed music given their durations and printed sizes. The algorithm is based on the two observations that the ideal spacing of notes is a logarithmic function of their durations, and that in the presence of several voices, the note with the shortest duration defines the space used for all simultaneous notes. The algorithm improves on previous ones in handling triplets and other complex rhythms simply and without special cases. It also allows for stretching and shrinking of lines of music for optimal line breaking and accomodating wide notes, maintaining the logarithmic spacing at all times. The results of the algorithm have been compared with good manual spacing of music, and the two are indistinguishable in the vast majority of cases.

Introduction

The conventions used for spacing notes, chords, and other symbols in printed music are very consistent, and they are also well understood in the sense that trained musicians can immediately recognize poor spacing. There is less agreement, however, on the best algorithm to use to produce good spacing, and documented attempts at this are incomplete and fraught with special cases. This paper proposes a simple rule that handles the vast majority of music spacing problems, or punctuation as musicians call it, consistently and correctly.

The goal of all varieties of automated document formatting is to relieve authors or editors of the burden of typographical detail, allowing them to concentrate on the content of the document. The MusiCopy project at the Ohio State University [5] is no exception, its goal being the production of publication-quality printed music from input consisting primarily of musical information, such as the pitches and durations of notes. Unlike text formatting, the spacing of symbols in music depends little on the physical sizes of the symbols, but instead reflects the durations and overlaps of the sounds it represents. In making spacing decisions, therefore, durations are the primary data with which we must work.

Musicians, however, almost never think about music as an unstructured collection of sounds, preferring to organize the sounds into *voices*, (often corresponding to instruments,) which are sequences of sounds and silences that do not overlap each other. Several voices performed in parallel then comprise a piece. For the purposes of this paper all elements of a voice will be called *notes*, regardless of the fact that they might, in fact, be chords or even rests. We assume that descriptions

* This work was partially supported by the National Science Foundation under grant number IST-8514308 and The Ohio State University. This paper is in final form and no version of it will be submitted for publication elsewhere.

of several parallel voices have already been preprocessed and collated, by software such as is described by Parrish and Gourlay [4], into a sequence of *simultaneities*, or *sims* for short, that record successive events in the piece of music. Each sim contains the time until the next sim, as well as information from each voice about notes that begin at that moment.

For the purpose of this paper, a sim contains four pieces of information for each voice: (1) a flag indicating whether or not there is a note beginning at this time in this voice, (2) the duration of the note if there is one, (3) the *left width* of the symbol, and (4) the *right width* of the symbol. The left and right widths can be understood best by looking at Figure 1, in which three notes of various sorts are illustrated. The space between notes for musical purposes is not measured as in text from the right edge of one note to the left edge of the next, but instead from the *beatline* of one note to the beatline of the next. The beatlines of notes, labelled B_i in the figure, generally fall at the left edge of the noteheads. Note that all three of the printed notes protrude to the right of their beatlines, to R_i, and two of the notes protrude to the left of their beatlines, to L_i. These protrusions are the notes' left and right widths, respectively. Some amount of stretchability and shrinkability are present in each space for justifying lines, but under no condition is the right width of one note allowed to overlap the left width of the following note.

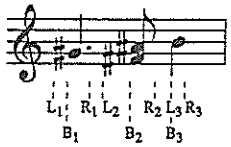

Figure 1. Beatlines and left and right widths of three notes.

The durations of notes are conventionally denominated in fractions of a "whole note," and we retain that convention here, using $\frac{1}{2}$ as the duration of a half note, $\frac{1}{4}$ as the duration of a quarter note, etc. The same unit of measure is used for the times between sims. The left and right widths of notes, for scale-independence, are given in units equal to the width of a notehead, or *notehead widths.*

Just as we are assuming that preprocessing has occurred to create input of precisely the correct form, we also limit the scope of this paper by omitting discussion of line breaking, an interesting and difficult problem in its own right. The spacing algorithm discussed here produces information appropriate for the line breaking algorithm described by Hegazy and Gourlay [3]. Specifically, for each sim it produces an *ideal width*, a *stretchability*, a *shrinkability*, and a *blocking width*. The ideal width is the proper distance from this sim's beatline to the next based on durations alone. The stretchability and shrinkability state the amount by which this space should be allowed to stretch or shrink in proportion to other sim spaces. These are used in justifying a line of music after line breaking, or in making room for particularly large symbols. Finally, the blocking width is the minimum allowable distance from this beatline to the next that will avoid the physical overlap of symbols.

Finding Ideal Widths in an Isolated Voice

The simplest case to consider is that of spacing a single voice. Input to the algorithm, under these conditions, simplifies to nothing more than a series of dura-

tions, one for each note in the voice. It is easy to see in printed music that longer notes occupy more space than shorter notes, and it is tempting to guess that notes receive space in direct proportion to their durations. This is not the case, however, and longer notes occupy proportionally less space than one would first expect. Figure 2 illustrates this, showing typical ideal widths used in printed music for notes of several durations. Tables similar to this appear in books on music notation, (e. g., Ross [6]), and also in papers on automated music formatting (Gomberg [2].) Gomberg implements his spacing by table look up, but this is not entirely satisfactory, because it provides no guidance in interpolating the space for durations not in the table, such as dotted notes or triplets, nor does it help in understanding how to stretch or shrink spaces when justifying lines.

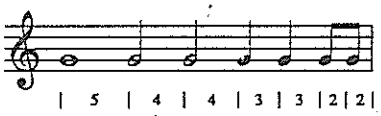

Figure 2. Typical spacings of various notes (in notehead widths.)

It is natural to look for a continuous function approximating these ideal spaces. Byrd [1], Tatum [7], and Vendome [8] have all independently proposed functions of the form $ab^{\log_2 d}$, where d is the duration of the note to be spaced, and a and b are constants chosen differently by the different people. This says that doubling the duration of a note increases its space by a factor of b, a value typically set near 1.5. Unfortunately, this kind of function will inevitably produce spacings much too large for very long notes if the constants are set to behave well for short notes.

Actually, there is a simpler function, $\log_2 d + 5$, that produces exactly the spacings prescribed by Gomberg and shown in Figure 2. In addition, generalized to $\log_2 d + k$, this function also accounts for spacings in the presence of shorter notes, or stretched or squeezed lines. For example, in the presence of sixteenth notes, or in a severely stretched line, the spacings of 5, 4, 3, and 2 in Figure 2 would change to 6, 5, 4, and 3, respectively, and $\log_2 d+6$ defines these spacings exactly. A remarkable fact is that all the stretchabilities and shrinkabilities of the sim spaces in the single-voice case are equal. In the example just given, all the spaces were stretched by one notehead width, and in general, stretching and shrinking are accomplished simply by modifying k.

All that remains to be specified, then, is the rule for choosing k. This introduces the concept of a *neighborhood*, a series of notes that should receive consistent spacing. More will be said about this in the last section, but we have found that a measure forms an adequate neighborhood nearly all of the time. We also observe that the shortest notes in a measure, if they are eighth notes or shorter, receive two notehead widths of space. If the shortest note is longer than an eighth note, it is spaced as if eighth notes are present in the measure.

The algorithm, then, is, for each measure, find the minimum duration m, compute $k = 2 - \log_2 \min\{m, \frac{1}{8}\}$, and then from the duration d_i of each note i in the measure, compute the note's ideal width $w_i = \log_2 d_i + k$. Settings for stretch and shrink appropriate for the Hegazy-Gourlay line-breaking algorithm turn out to be 1 and $\frac{1}{2}$, respectively, reflecting the fact that it is more acceptable to stretch a line of music than it is to shrink it.

Spacing Several Simultaneous Voices

The rule that allows us to generalize this algorithm to more than one voice is simply that the note with the shortest duration defines the space used by all simultaneous notes. In the first two measures of Figure 3, the lower voice is spaced exactly as it would be in isolation, while the upper voice, consisting entirely of longer-duration notes, is distorted to maintain the required vertical alignment of notes with simultaneous attack times. For example, in the first measure the second half note begins at the same time as the third quarter note, and so their beatlines coincide in spite of the fact that the half note is farther right than the isolated spacing of Figure 2 would suggest. In the second measure each of the three "triplet" half notes in the lower voice have two-thirds the duration of the ordinary half notes in the upper voice. These notes are spaced appropriately for their durations, and again, the upper voice is stretched to coincide. The difference in this case is that the second note in the upper voice begins half way through the duration of the second note in the lower voice, and so the beatline of the upper note bisects the space of the lower note.

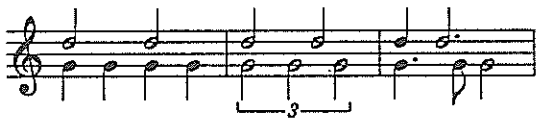

Figure 3. Spacing of two voices.

The third measure is more complex because initially the upper voice has the shortest note, but later the lower voice has it. For this reason, neither voice is spaced exactly like it would be in isolation. The same spacing rule applies, nevertheless. The first sim is given the three spaces appropriate to the quarter note in the upper voice. The second sim, which marks the beginning of the dotted half note in the upper voice ($d = \frac{3}{4}$.) occurs before the end of the dotted quarter note in the lower voice ($d = \frac{3}{8}$.) Since the dotted quarter note is shorter, it defines the space for this sim. A complete dotted quarter note would receive $\log_2 \frac{3}{8} + 5 \approx 3.6$ notehead widths, but only one-third of the duration of this note remains before the next sim, the eighth note. Thus, the second sim receives 1.2 notehead widths of space. After this, the lower voice retains the shortest duration and therefore controls the spacing. The significant observation here is that sims spanning fractions of notes should be handled literally as fractional notes, and not as full notes of short duration. This allows us to handle all three cases in Figure 3 in the same way, instead of treating them as separate special cases as is done by Gomberg [2] and Byrd [1].

The general procedure, then, is first to find k as before, based on the minimum duration of all the notes in all the voices in the measure. Then, for each sim i, find d_i, the shortest duration among all the notes starting or continuing at the time of the sim. Also, find f_i, the fraction of this shortest note that will elapse before the next sim. Finally, find the ideal width of the sim $w_i = f_i(\log_2 d_i + k)$. We want the stretch and shrink for each sim to be appropriate for the note controlling its spacing, and so we set them to f_i and $f_i/2$, respectively.

Accomodating Large Note Widths

Thus far we have treated notes as if they had no physical widths when printed. In fact, they sometimes have very substantial widths that force them farther apart than ideal, as is illustrated in Figure 4. In addition to the ideal width, we need to compute a blocking width for each sim that will indicate the closest that it should be allowed to come to the sim following. In the first measure of the figure, the sharps take up 2.25 notehead widths, and the second eighth note takes up another notehead width. The second sim, therefore, has a blocking width of 3.25, which requires extra space because it is greater than the ideal width of 2. The general case is complicated, however, by the fact that blocks can span several sims, as in the second measure. As before, we have two sharps attached to the third sim, but the left width belongs to the upper voice rather than the lower one. As before, there is 3.25 notehead widths of blocking, but the upper voice is absent in the second sim and the block is now between the first and third sims. Ideal spacing puts these sims 4 notehead widths apart, and so the block does not require any extra space.

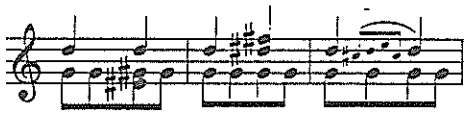

Figure 4. The effect of extra width.

The third measure shows a more extreme case where the width of the block exceeds even the large ideal space in the upper voice. The grace notes require a left width of 5 in the second note of the upper voice. Added to the right width of the first note, the size of the block is 6. This requires extra space, but the block spans two sims, so the blocking width must somehow be distributed over both sims. If we think of the grace notes as stretching a small neighborhood, it is clear that we should maintain the logarithmic spacing rule within the neighborhood. Thus, we should distribute the excess width among sims in proportion to their stretchability, in this case giving both the first and second sims blocking widths of 3.

A convenient way to add the computation of blocking widths to the evolving spacing algorithm is to add a step, prior to the ideal width computation, to look back at preceding sims and to increase their blocking widths, if necessary, to allow for the left widths of the current sim. We will usually have to look back only one or two sims, and never farther than the beginning of the measure, so this approach is quite tractable.

For each voice that has a note beginning in the current sim s, we need to do the following: First, find p, the sim with the last occurrence of a note in this voice, and find $z = r_p + l_s$, the total block in this voice, the sum of the right width of the previous note and the left width of the current one. Then, make a pass over the sims from p to $s-1$ and find $k' = (z - \sum_{i=p}^{s-1} w_i) / \sum_{i=p}^{s-1} f_i$, the increment to k that will stretch the spanned sims to a total width of z. Finally, go back over each of the spanned sims i, and replace its blocking width b_i with $k' f_i + w_i$, if the latter is larger.

Results and Remaining Problems

The algorithm as described is essentially complete. The implementation contains a number of additional technical details for handling barlines (which act as if they have small durations, but do not belong to neighborhoods), and for handling symbols without duration such as tempo marks and dynamics (which participate in blocking width computation, but not in ideal space computation.) Also, care must be taken in correctly specifying the input for whole notes and whole rests (which can be printed well to the right of their beatlines.) Nevertheless, there are no important new ideas incorporated in these features.

The algorithm has been tested on numerous small examples of music and on several extended pieces with excellent results. In a typical piece with 50 measures, the automatic spacing and subsequent line breaking [3] was indistinguishable from the original hand spacing in all but one measure. Both forms of the line containing this measure are shown in Figure 5, and we can see that the automatic spacing gave noticeably more space to the longer notes of the penultimate measure. The manual spacing is a little bit better, and the problem turns out to be a subtle question of neighborhoods—the copyist wanted all of the eighth notes in the line to receive consistent spacing, and noticed as well that there were no sixteenth notes in the measure in question. He then made the judgement that the line as a whole would look better if he violated conventional spacing for the one measure by spacing it as if the thirty-second notes were sixteenth notes. While with hindsight this particular pattern of reasoning could conceivably be built into the spacing algorithm, this general level of human judgement is well beyond the scope of any formatting system, and we are gratified that it is only at this level that our algorithm deviates from manual spacing.

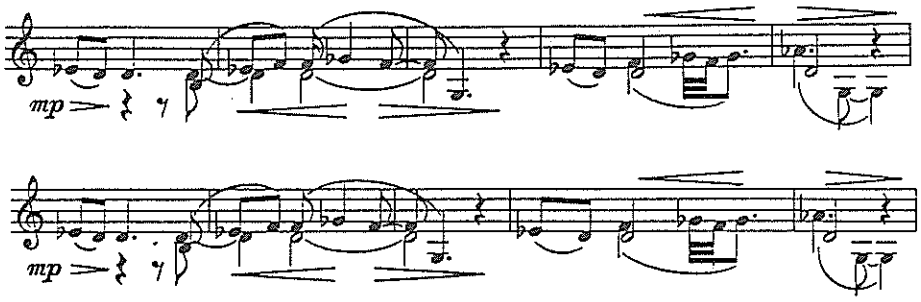

Figure 5. Worst-case comparison of a line set manually (above) and automatically (below.)

One way to mitigate this and similar neighborhood problems is to provide a mechanism by which the user of the music formatting system could explicitly set the bounds and parameters of neighborhoods. Not only should one be able to assign a particular value to k, but one should also have access to the base of the logarithms used. Using logarithms base 4 would produce almost precisely the effect the copyist desired in Figure 5.

Another rare problem that is not adequately addressed by this algorithm is that of "crossed voices," i. e., two voices on one staff in which notes of the lower voice are higher on the staff than notes of the upper voice. If the crossed notes

are simultaneous, they will be assembled properly by our preprocessor [4] into a single sim, such as the last sim of the first measure of Figure 5. If the crossed notes are not quite simultaneous, however, for example if the voices were crossed in the second or third measures of Figure 4, our spacing algorithm would simply allow the symbols to overlap. Again, the problem could be mitigated by giving the user the ability to deliberately misstate left and right widths, forcing things farther apart, but a challenging future problem would be to generalize this algorithm to space crossed voices properly on the first try.

Perhaps the empirical success and relative simplicity of this spacing algorithm are enough to encourage its acceptance. There is an intriguing possibility, however, that it reflects more than simple musical convention, and says something about visual perception. It is well known that intensities of many stimuli are perceived logarithmically, and that the retina has neurons that measure changes in stimulus as well as absolute levels. The spacing algorithm described here suggests that printed music captures an inherent logarithmic encoding of duration, and that musicians respond not to the absolute spacing of notes, but to the differences in spacing of successive notes. This conjecture lends some psychological plausibility to the spacing algorithm, but confirming it would require work well beyond the scopes of music and computer science.

Bibliography

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

[2] Gomberg, David A., "A Computer-Oriented System for Music Printing," *Computers and the Humanities*, vol. 11, pp. 63–80, 1977.

[3] Hegazy, Wael A., and Gourlay, John S., "Optimal Line Breaking in Music," Ohio State University, Dept. of Computer and Info. Science, OSU-CISRC-10/87-TR33, 1987.

[4] Parrish, Allen, and Gourlay, John S., "Computer Formatting of Musical Simultaneities," Ohio State University, Dept. of Computer and Info. Science, OSU-CISRC-10/87-TR28, 1987.

[5] Parrish, Allen, *et al.*, "MusiCopy: An Automated Music Formatting System," Ohio State University, Dept. of Computer and Info. Science, OSU-CISRC-10/87-TR29, 1987.

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

[7] Tatem, J. E., *et al.*, "ENGRAVE—An Expert System that Plans the Spatial Layout of Music," unpublished manuscript, 1985.

[8] Vendome, Richard, untitled, unpublished manuscript concerning the Oxford Music Processor, 1986.

Acknowledgements

I would like to thank Dean K. Roush for his expertise in music notation and for his permission to use excerpts from his composition "Lacrimosa" in this paper. I also thank Yiling Tien for her preliminary work on this problem.