An Improved Algorithm for Spacing a Line of Music

Kai Renz
Darmstadt University of Technology
Wilhelminenstr. 7, D-64283 Darmstadt, Germany
email: renz@iti.informatik.tu-darmstadt.de
http://www.noteserver.org/

Abstract

Automatically spacing a line of music has been discussed various times in the past. All of the so far published algorithms are variations of a scheme originally published by Gourlay in 1987. In general, Gourlay's algorithm produces nicely spaced lines of music. However, the algorithm inherently fails to give good results in certain situations, where equal spacing of equal note durations is desired. A number of examples for these cases are presented in this paper. A solution to the problem is presented, which improves the existing model and leads to an improved algorithm for spacing a line of music that very closely resembles human spacing.

The improved algorithm for spacing is a three step process, which is based upon Gourlay's original work: first, regions of notes with equal duration are determined in the score. Then, for each such region it is checked, whether the original Gourlay algorithm calculates spacing errors. If this is the case, an alternate spacing is calculated and applied, if certain tolerance conditions are met.

The paper includes various examples comparing the different spacing algorithms and shows how the improved algorithm handles spacing better than all of the most popular existing notation systems.

1 Introduction

Traditionally, musical information is transmitted using scores. Today, quite a number of music notation systems are in use, most of which include expert knowledge of conventional music notation. One of the most important properties of conventional music notation is the strong coupling of graphical appearance with the underlying musical semantics. A well written score is easy to read and perform, because the intended musical ideas are clearly visible. One aspect of good music notation concerns the connection between note duration and the associated white space: a long note is followed by more space than a short note. The process of automatically determining the amount of white space in between notes (and rests) is called spacing. Spacing a line of music has been discussed quite often in the past [Byr84, Gou87, BH91, HB95, Gie01]. The most important contribution to the problem was given by Gourlay in 1987. His spacing model, which in turn was inspired by Donald Knuth's work on TEX [KP81], seemingly builds the basis for a large number of the currently implemented notation systems [1]. His model can be described as a Spring-Rod-Model [2]: a line of music is interpreted as a collection of springs with some spring constants on which a force is exerted to stretch or shrink the line to a desired length. Rods are introduced to ensure that noteheads and musical markup (like for example accidentals) do not collide, especially when spacing is tight. The rods are used to pre-stretch the springs to have a guaranteed minimal extent. In contrast to formatting text, the issues found in formatting music are more complex. Because simultaneous voices must be vertically aligned, a spacing algorithm must be capable of handling overlapping note durations in different voices while still maintaining a spacing that closely follows individual note durations.

All of the subsequently published work on spacing a line of music can be described as minor variations of Gourlay's original model. It was therefore somewhat surprising to detect that Gourlay's algorithm (and therefore all of its direct descendants) fails to produce good spacing in certain situations. The reason for these deficiencies stems from the fact, that sequential individual notes with equal duration might be spaced unequally because of simul-taneous notes in other voices. As stated above, the coupling of note duration and spacing is very strong, therefore notes with equal duration should be spaced equally whenever possible. Because these cases do occur in real scores (and also in computer

[1] Obviously, it is not really possible to detect the implemented spacing algorithm for software that is distributed without the source code.

[2] The original article by Gourlay does not use the word "springs" but speaks of boxes and glue, which in turn was inspired by TEX. We find that the term "spring" more closely corresponds to the physical model being used.

generated music), a solution to the problem was searched for and found. This solution led to an improved algorithm for optimally spacing a line of music that automatically detects and corrects the above mentioned spacing errors. First, regions of notes with equal duration are determined in the score. Then, for each such region it is checked, whether the original Gourlay algorithm creates spacing errors. If this is the case, an alternate spacing is calculated and applied, if certain tolerance conditions are met.

Figure 1 shows, how the improved spacing algorithm (b) compares to Gourlay's original algorithm (a): The spacing of the triplet in the first voice is exactly equal when the improved algorithm is used. This effect is highly desirable, because the graphical appearance of the triplet now directly conveys that each individual triplet note has equal duration.

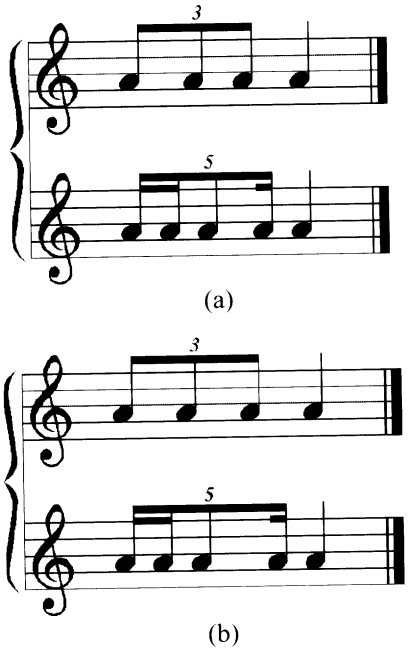


Figure 1: Gourlays' original algorithm (a) compared to the improved spacing algorithm (b).

The improved algorithm has been implemented in the GUIDO Notation Renderer, which is used to create conventional scores from GUIDO descriptions [Ren00, HHRK98]. Because the GUIDO Notation Renderer is also used in the NoteServer [RH98], the improved algorithm can be directly examined online[3].

The rest of this paper is structured as follows: first, an overview of Gourlay's algorithm for spacing a line of music is given. Another publication on spacing and some music notation systems using Gourlay's algorithm are shortly mentioned. We then show how Gourlay's algorithm fails to space even a simple example of a rhythmically complex section. Finally, the improved algorithm is introduced and it is demonstrated, how the problematic cases are then spaced in an optimal way. Additionally, a passage from a real score is spaced using the original and the improved algorithm, which closely resembles the spacing of a human engraver. The conclusion discusses some issues that could still be improved in the presented solution.

[3] The NoteServer is a free online service, which creates music notation from GUIDO descriptions. It is available at http://www.noteserver.org

2 Gourlay's Algorithm for Spacing

This section gives a brief overview of Gourlay's algorithm for spacing a line of music [Gou87] and shortly discusses additional publications and notation systems based on it. Obviously, some of the details of Gourlay's algorithm are omitted to focus on the relevant issues being improved later.

The basic idea of Gourlay's algorithm can be summarized as follows: a single line of a score is interpreted as a list of onset times, which is sorted by time position; an onset time occurs whenever a note or rest begins in one of the voices. In order to determine the correct amount of space to put in between the onset times, springs are inserted between successive onset times. In order to stretch (or shrink) the line to its desired length, a force is exerted on the whole collection of springs. Each spring is then stretched (or shrunk) depending on the force and the spring constant, which mainly depends on the spring duration[4] and on the duration of notes (or rests) present at the time position of the spring. Hooke's Law describes, how a force F, an extent x and a spring constant c are related: $F = c \cdot x$. To avoid collisions of noteheads, accidentals, and other musical markup, rods are introduced, which determine the minimum stretch for one or more springs.

Calculation of spring constants

The calculation of spring constants in Gourlay's algorithm, which will be modified by the improved algorithm, needs to be examined more closely. Basically, the space placed after a note (or rest) solely depends on the duration of the event: there is generally less space placed after a sixteenth note than after a whole note. A suitable way to calculate the actual amount of space for a given duration d is given by the formula

$$\begin{gathered} \mathrm{space}(d) = \psi(d) \cdot \mathrm{space}(d_{\min}) \\ \text{where} \\ \psi(d) = 1 + a \cdot \log_2\left(\frac{d}{d_{\min}}\right) \end{gathered} \tag{1}$$

where a is some constant (usually in the range of 0.4 and 0.6), and $d_{\min}$ is some smallest duration for which the required space is predefined. In current notation software (like, for instance in Finale, LilyPond, etc.)

[4] The term "spring duration" is defined exactly below.

quite a number of minor variations of this scheme are actually implemented (see for example [Gie01]), but the general principle of using a logarithmic function for calculating the required space is agreed upon widely. Tweaking the parameters for a and $d_{\min}$ results in slightly different spacing and it is usually a matter of personal (or publishers) taste, which values to use.

The formula for $\psi(d)$ from equation (1) is also used to determine the spring constants for a given line of a score: Let s be a spring between two successive onset times o_1 and o_2. The resulting spring duration d_s is simply d_s = timeposition(o_2) - timeposition(o_1). Let d_i be the shortest duration of all notes (or rests) beginning or continuing at timeposition(o_1). Then the spring constant c is calculated as

$$c = \frac{d_i}{d_s} \cdot \frac{1}{\psi(d_i) \cdot \mathbf{space}(d_{\min})} \qquad (2)$$

Gourlay introduced this formula in order to get optimal spacing even when rhythmically complex structures (like tuplets versus triplets) are present.

As indicated above, a number of variations of Gourlay's algorithm have been proposed. The newest publication on spacing a line of music is by Haken and Blostein [HB95]. They were the first to actually use the terms springs and rods. Even though their terminology is different, the general ideas of Gourlay are present. The only significant difference lies in the handling of rods, which are called blocking widths by Gourlay. Haken/Blostein introduced an elegant way to pre-stretch the springs using a two-step process: first, only those rods are considered that only stretch one spring. Then the remaining rods (which all span more than one spring) are checked. In this phase, the required force to stretch a chain of springs to the desired rod length is calculated. The rod requiring the *maximum force* is applied first; other rods stretching the same springs can then be discarded. This procedure is more efficient than Gourlay's original algorithm, which just iterates through all rods without pre-sorting.

GNU LilyPond, a musical typesetting system based on TEX [NN01], is also using Gourlay's algorithm for spacing a line of music. One interesting detail of LilyPonds spacing is the fact that the value for $d_{\min}$ in equation (1) is determined separately for each measure. This sometimes leads to an uneven spacing of measures, which might be musically misleading as shown in Figure 2, where the general rhythm of measures one and two is quite similar, but the spacing is very uneven.[5]

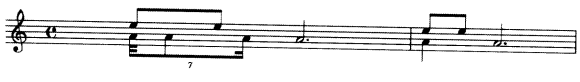


Figure 2: LilyPond Spacing Problem

[5] Obviously, it can be argued, whether a spacing like this is desirable or not.

The spacing algorithms used by commercial notation software are usually not disclosed: therefore, a small number of spacing tests were performed with the two most popular systems Finale and Sibelius[6]. In the case of Finale, it seems like some variation of Gourlay's algorithm is used: the obtained spacing for rhythmically complex pieces is generally very good. Sibelius, on the other hand, created awkward spacing even for some rather simple examples. It seems like the used algorithm does not allow noteheads to overlap, even if they are in completely independent voices. Figure 2 shows, how Sibelius fails to space a rhythmically complex measure in an acceptable way.

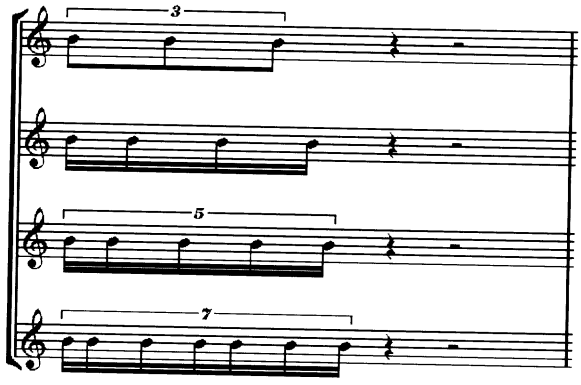
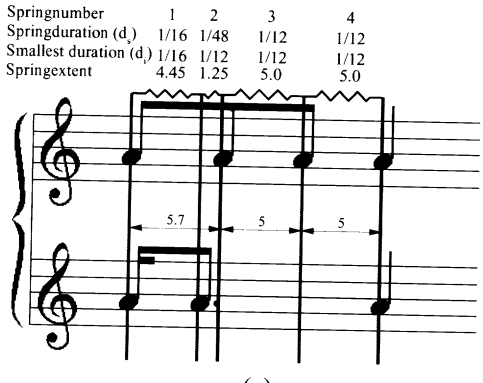


Figure 3: Spacing in Sibelius

3 Spacing Problems in Gourlay's Algorithm

Gourlay's algorithm for spacing a line of music calculates good spacing on a large body of musical scores. Nevertheless, the original algorithm creates non-optimal spacing in some cases. Figure 1 (a) shows, how Gourlay's algorithm spaces a short two voice piece. It should be noted, how the individual notes of the triplet in the upper voice are spaced differently: the space between the first and the second note of the triplet is significantly greater than between the second and the third note, even though their duration is equal. We call this error "a neighborhood spacing error ".



(a)

[6] For our tests we used Finale 2001b and Sibelius 1.4

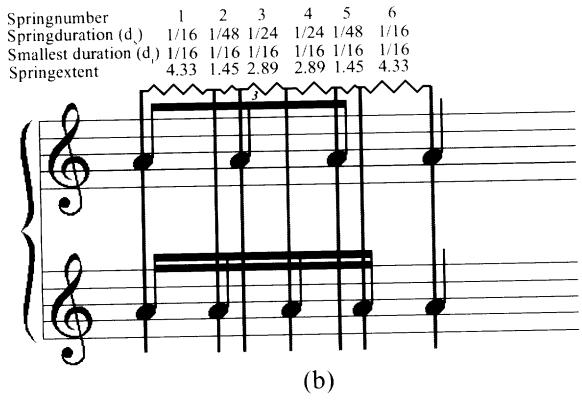


Figure 4: Spacing of Gourlays' algorithm including springs.

The reason for the unequal spacing can be seen rather easily: as shown in Figure 4 (a) the first triplet note spans two springs (s_1 and s_2) with spring constants $c_1 = \frac{\frac{1}{16}}{\frac{1}{16}} \cdot \frac{1}{\psi(\frac{1}{16})}$ and $c_2 = \frac{\frac{1}{12}}{\frac{1}{48}} \cdot \frac{1}{\psi(\frac{1}{12})} = \frac{3}{\psi(\frac{1}{12})}$. When combining s_1 and s_2 into a combined spring $s_{1,2}$ the spring constants $c_{1,2}$ is calculated as

$$c_{1,2} = \frac{1}{\frac{1}{c_1} + \frac{1}{c_2}} = \frac{1}{\psi(\frac{1}{16}) + \frac{1}{3}\psi(\frac{1}{12})}$$

It is obvious, that the spring constant $c_{1,2}$ is not equal to $c_3 = c_4 = \frac{1}{\psi(\frac{1}{12})}$. Therefore, when a force F stretches the system, the springs s_1 and s_2 are stretched to the extent $x_{1,2} = \frac{F}{c} = F \cdot (\psi(\frac{1}{16}) + \frac{1}{3}\psi(\frac{1}{12}))$ and the springs s_3 and s_4 are both stretched equally to $x_3 = x_4 = F \cdot \psi(\frac{1}{12})$. Clearly, $x_{1,2}$ and x_3 (and x_4) are not equal.

To understand, why this error does not occur in the standard 3 against 4 case (see Figure 4 (b)), one must understand how the fraction $\frac{d_c}{d_c}$ in equation (2) is supposed to work: consider Figure 4 (b), which differs from (a) by replacing the second voice with four sixteenth notes. Now, the value of d_i for each of the springs is $\frac{1}{16}$, so that each spring is stretched as a partial of a sixteenth-note spring. To clarify this, consider springs s_2 and s_3 of Figure 4 (b). The respective spring-constants are $c_2 = \frac{\frac{1}{16}}{\frac{1}{48}} \cdot \frac{1}{\psi(\frac{1}{16})} = \frac{3}{1} \cdot \frac{1}{\psi(\frac{1}{16})}$ and $c_3 = \frac{\frac{1}{16}}{\frac{1}{24}} \cdot \frac{1}{\psi(\frac{1}{16})} = \frac{3}{2} \cdot \frac{1}{\psi(\frac{1}{16})}$. When combining springs s_2 and s_3 the springs constant is

$$c_{2,3} = \frac{1}{\frac{1}{3}\psi(\frac{1}{16}) + \frac{2}{3}\psi(\frac{1}{16})} = \frac{1}{\psi(\frac{1}{16})}$$

The result is, that the springs s_2 and s_3 are stretched so that there sequential combination behaves just as a single spring for one sixteenth note. The same is true for springs 4 and 5.

One other spacing problem of Gourlay's algorithm concerns lines of music that are spaced very loosely. Consider Figure 5, where a single line of music has been stretched rather heavily. In this case, Gourlay's algorithm (a) results in unequal spacing of the eighth-notes in the second voice: because of the 32nd notes in the first voice, the first eighth-note of the second voice is followed by much more space than the following ones, which are all spaced equally. The improved algorithm (b) distributes the space evenly.

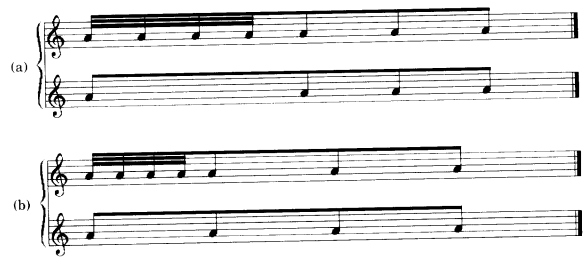


Figure 5: Loose Spacing: Gourlay algorithm (a) and improved algorithm (b)

4 An Improved Spacing Algorithm

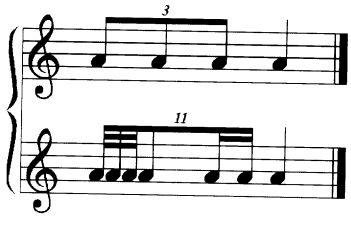
It can be deduced from the last section that, for solving the neighborhood spacing problem, an algorithm must automatically detect those regions of springs, where spacing errors might occur and then set an appropriate value for d_i only for the relevant regions. This is exactly what our proposed improved spacing algorithm does, which is now described in detail:

1. The original algorithm of Gourlay for the given line of music is carried out. For each spring, the value $1/d_i$ from equation (2) is saved, instead of directly calculating and saving the spring constants.
2. For each voice the location of neighborhoods of notes with the same duration are determined.
3. Each of the calculated neighborhoods from step (2) is checked for neighborhood spacing errors. This is done by averaging over the values d_i for each spring covering a note in the neighborhood. If the average value differs for any note of the neighborhood, a spacing error is detected.
4. For all error regions of step (3), it is checked, whether an other neighborhood reaches into it; if this is the case, then the boundary of the error-region is extended to also cover the new neighborhood. This step is necessary so that the fixing a spacing error in one of the error-regions does not introduce a new spacing error in another voice.
5. For the finally determined error-regions of step (4), three spring constants are calculated: the spring constant for the original Gourlay algorithm, the spring constant for using an average of the d_i, and a spring constant which uses the minimum d_i for the whole region.
6. The difference of the original Gourlay constant and the average constant is determined. If it is

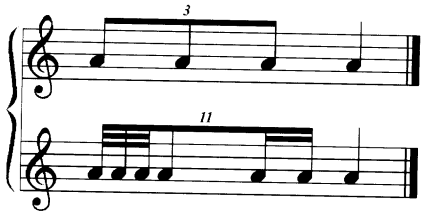
within a tolerance band (less than 0.05), then the average value for d_i is taken for the region. Otherwise, the difference of the original constant and the minimum d_i-constant is calculated. If this difference is within another tolerance band (less than 0.17), then the minimum duration of the region is chosen as d_i. Otherwise the original Gourlay value for d_i is chosen for the region.

7. Before a line is finally spaced, an additional step is performed to avoid the wrong spacing of loosely spaced lines. This is done by determining, whether any note with the smallest note duration of the whole line is followed by more than two noteheads of white space using the calculated spring constant. If this is the case, the smallest note duration for the whole line is taken as a value for d_i and the spring constants for the line are recalculated accordingly.[7]

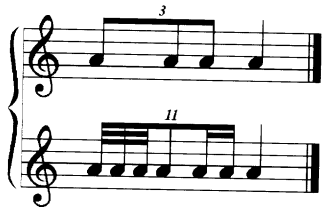
The values for the tolerance bands for step 6 of the algorithm have been chosen through various experiments. Figure 6 shows an example for taking the average (a), the minimum (b) and the original Gourlay value (c). Although the neighborhood spacing error in the first voice is removed in cases (a) and (b), choosing the average value results in too little space for the short notes in the beginning of the second voice. Choosing the minimum duration (b) results in a spacing, which is too wide. In this example, the original Gourlay algorithm (c) gives the best result, although the neighborhood spacing error in the first voice is clearly visible. The tolerance bands are indications, when to tolerate a neighborhood spacing error so that the overall spacing remains acceptable.



(a)



(b)



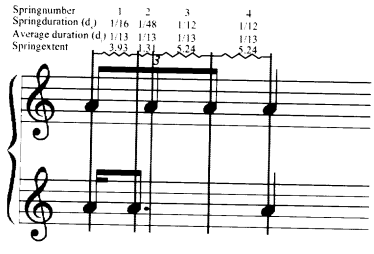
(c)

Figure 6: Showing the effect of choosing different relative duration values.

To see, how the algorithm solves the neighborhood spacing problem of the last section (see Figure 4 (a)), we give a trace of the improved algorithm.

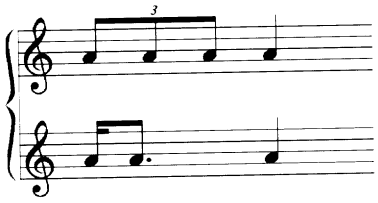
1. The original Gourlay algorithm is used to determine the values 1/4 for each spring: 16, 12, 12, 12. These values can be found in Figure 4 (a).
2. The first voice has a neighborhood list of (1, 3, 4, 5) which means, that the notes covering springs 1 up to 3 (not including the last value) and 3 to 4 and 4 to 5 are successive notes having the same duration (in our example this is 1/12). There is no neighborhood list for the second voice.
3. The neighborhood list (1, 3, 4, 5) is checked for spacing errors: the average for the first note (going from springs 1 up to 3) is (16+12)/2 = 14, the average for the second and third note is both 12. Therefore, a spacing error is detected.
4. Because there are no more neighborhoods, the error region reaches from spring 1 up until spring 5.
5. The three spring-constants are calculated: $s_{\text{gourlay}} = 1.81665$, $s_{\text{minimum}} = 1.60325$, $s_{\text{average}} = 1.81631$.
6. The difference $|s_{\text{gourlay}} - s_{\text{average}}| = 0.00034$ is within the tolerance band. Therefore, the average value for $1/d_i = (16+12+12+12)/4 = 13$ is chosen. Spacing is done, as if the smallest duration is a 13th note.

The resulting spacing can be seen in Figure 7 (a) and (b), which also shows a comparison to the old Gourlay algorithm (c).

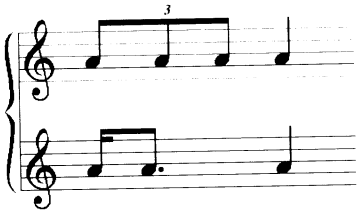


(a) Improved algorithm with springs

[7] This step was performed to obtain the result of Figure 5 (b).



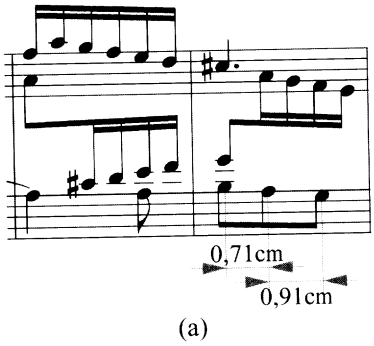
(b) Improved algorithm



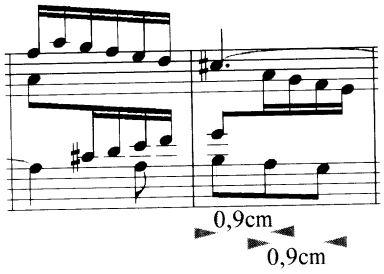
(c) Gourlay algorithm

Figure 7: Correction of neighborhood spacing errors by the improved algorithm

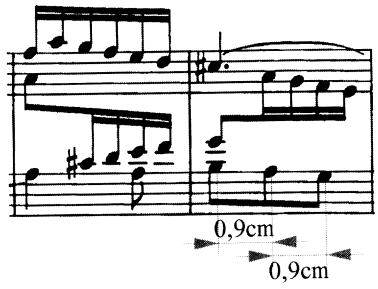
To show that the algorithm not only concerns somewhat constructed examples, Figure 8 shows a two-measure excerpt from a Fugue by J.S.Bach (BWV 856) as it is spaced by Finale (a), the improved algorithm (b) and in the Henle Urtext edition (c), which has been spaced by an expert human engraver [vI70]. It is clearly visible, that (b) and (c) strongly emphasize on the equal spacing of the eighth-notes in the bass voice of the second measure, whereas the spacing algorithm of Finale[8] spaces the first eighth note tighter than the following two. Of course it can be argued, whether the spacing of (b) and (c) is better than the spacing in (a); it is surely a matter of personal taste and also of musical semantics which spacing is preferred. The point made here is that any spacing algorithm should be adjustable to accommodate personal preferences.



(a)



(b)



(c)

Figure 8: Bach Fugue BWV 856: Finale (a), improved algorithm (b), hand engraver (c)

5 Conclusion

An improved algorithm for spacing a line of music was presented. It enhances Gourlay's original algorithm by taking the equal spacing of neighboring notes with the same duration into account. The improved algorithm spaces some instances of complex rhythms better than any of the existing most used notation systems. The algorithm can nevertheless be implemented easily. While the results are very promising, some parts of the algorithm might further be improved: first, the detection of neighborhoods could be extended to also cover sequential equal note durations in different voices, not only for each voice independently. Second, the tolerance bands could be further refined. One easy solution for determining regions for equal spacing would be to explicitly ask the user to provide this information.

The automatic generation of conventional scores still remains to be a complex problem. Although many improvements in automatic notation systems have been made, quite some time is still required to obtain good results in any possible perceivable situation.

References

[BH91] Dorothea Blostein and Lippold Haken. Justification of printed music. *Communications of the ACM*, 34(3):88–99, March 1991.

[Byr84] Don Byrd. *Music Notation by Computer*. PhD thesis, Department of Computer Science, Indiana University, 1984.

[8] Finale spaces this example similar to Gourlay's algorithm.

[Gie01] Martin Gieseking. *Code-basierte Generierung interaktiver Notengraphik*. PhD thesis, Universität Osnabrück, 2001.

[Gou87] John S. Gourlay. Spacing a Line of Music. Technical report, Ohio State, 1987.

[HB95] Lippold Haken and Dorothea Blostein. A New Algorithm for Horizontal Spacing of Printed Music. In *Proceedings of the 1995 International Computer Music Conference*, pages 118–119, Banff Centre for the Arts, Banff, Canada, 1995. International Computer Music Association.

[HHRK98] Holger H. Hoos, Keith A. Hamel, Kai Renz, and Jürgen Kilian. The GUIDO Notation Format A Novel Approach for Adequatley Representing Score-Level Music. In *Proceedings of the 1998 International Computer Music Conference*, pages 451–454, University of Michigan, Ann Arbor, Michigan, USA, 1998. International Computer Music Association.

[KP81] Donald E. Knuth and Michael F. Plass. Breaking paragraphs into lines. *Software – Practice and Experience*, 11(11):1119–1184, November 1981.

[NN01] Han-Wen Nienjuys and Jan Nieuwenhuizen. *GNU LilyPond – The music typesetter*, March 2001. The newest tutorial and reference manual available at the webpage and downloaded at January, 21st, 2002.

[Ren00] Kai Renz. Design and Implementation of a Platform Independent GUIDO Notation Engine. In *Proceedings of the 2000 International Computer Music Conference*, pages 469–472, Berlin, Germany, 2000. Berliner Kulturveranstaltungs GmbH, International Computer Music Association.

[RH98] Kai Renz and Holger H. Hoos. A WEB-based Approach to Music Notation using GUIDO. In *Proceedings of the 1998 International Computer Music Conference*, pages 455–458, University of Michigan, Ann Arbor, Michigan, USA, 1998. International Computer Music Associa-tion.

[vI70] Otto von Irmer, editor. *J. S. BACH – Das Wohltemperierte Klavier*, volume 1. G. Henle Verlag, München, 1970.