

Simulation of Bowing Decisions on a String Instrument

Bill Findlay & Cordelia Hall



Department of Computing Science
University of Glasgow
Glasgow G12 8QQ
Scotland

TR-1999-41
July 1999

For more information about Technical Reports
and Research at the

Department of Computing Science
University of Glasgow,

send mail to

reports@dcs.gla.ac.uk

or consult our Worldwide Web pages at

<http://www.dcs.gla.ac.uk/>

Technical Reports Coordinator
Department of Computing Science
University of Glasgow
Glasgow
G12 8QQ

phone +44 0141 330 4256
fax +44 0141 330 4913

Simulation of Bowing Decisions on a String Instrument

Bill Findlay, Cordelia Hall

Department of Computing Science,
University of Glasgow,
G12 8QQ,
Glasgow, UK
{wf, cvh}@dcs.gla.ac.uk

Abstract String players have long known that the hard part of playing their instrument is getting the bowing right. In particular, a bowing must be worked out very carefully if it is to tell them what part of the bow to use for each note. Such detailed knowledge is required if the performer is to practice properly. This paper describes an experiment in calculating the direction and proportion of bow used for each note in the first and second movements of the first violin solo part of Bach's double violin concerto. The program reports when the player has proposed an unworkable bowing. The player then fixes the problem and the program continues. We also discuss partial automation of the entire task as future work.

1. Introduction

It has long been said that string instruments such as the violin are hard to play at all, and very hard to play well. There are several reasons for this. Each hand is responsible for finding positions that are learned as the musician practices. In general, the left hand selects the pitch to be played and the right determines the sound of the note (although the left hand contributes vibrato and the right also selects the string to be stopped). Once the musician has learned to hear pitches accurately, it is easy to tell if the left hand is performing adequately. If it isn't, there are clever things one can do, like shifting the whole hand to the new position, then placing the finger, or making sure that shifts are done on half steps as much as possible.

However, the right hand is a real challenge. Players with perfect pitch get substantial help with their left hands because they can orient themselves rapidly. But becoming good at bowing requires that the player think very carefully about the nature of a good sound and the kind of bowing that entails. After working hard to mentally articulate the motions to be used when playing a piece at speed, the musician must practice so that these are the motions used in performance. Generally, they are found by trial and error; very often by the teacher rather than the student. There is no common terminology for the portion of bow used, other than the tip and the frog (the heaviest part of the bow). The musician may move the bow in one of two directions: towards the tip, which is called an upbow (<), and towards the frog, which is called a downbow (!). A *bowing* is a sequence of annotations indicating the direction in which the bow is expected to move as the marked passage is played.

The question of why there is little terminology is interesting. There are lots of words that describe various attacks, such as *martele*, or *spiccato*, and books on string playing give advice on how to perform these strokes. However, bowings only include annotations that indicate the direction in which the bow is to be moved; the string player must either recall where each stroke begins and ends or invent notation to preserve this information. The results of our experiments are preserved implicitly in the bowings we use. All of the hard won experience needed to develop a sequence of strokes that express a piece well is encapsulated laconically in the notation we use. We suspect many musicians forget where they should be on the bow at any one point because they don't have help in remembering. This may be damaging to professional players, because it can eventually become their duty to decide upon the bowings used by their section in a professional orchestra. In any case, they are required to be able to follow the bowings given by their section head, and to adjust to new bowings during rehearsals and performance. As soloists, they will often wish to create their own approach to their repertoire, which requires that they explore a large variety of bowings.

1.1 Previous work

Much work has been done on modeling the behaviour of the violin and bow [Schelleng 1973], and some measurements have been made of two professional violinists [Askenfelt 1989] as they played. Askenfelt concluded that the bow-bridge distance was of more importance than bow velocity, and that a high bow velocity soon becomes uncomfortable. Hall was taught (and confirmed this by evaluation) that the single most useful thing a string player can do is to learn how to use more bow. It is uncomfortable if the right arm is not ready for it, but the results are well worth changing the right arm motion. It is obvious that more bow helps in loud passages; and it also benefits soft sections. As observed by Carl Flesch [Flesch 1939, p. 93], 'Diminished bow expenditure is a comfortable, yet in general an inferior means of securing a lesser degree of tone power because the carrying power of the tone is noticeably diminished thereby'. At the very least, it is important to be conscious of the bow position so that a reasoned choice can be made. Previous work in this direction has been very encouraging. Shinichi Suzuki taught children to play by using tape to mark both the fingerboard and bow, so that they would concentrate their efforts on using the middle, the portion of bow that could be mastered quickly without much effort. Later, they were taught less systematically to use the tip and frog. And there have been a number of well-known teachers, such as Dorothy DeLay at Juilliard, who have emphasised exercises aimed at making the player as conscious as possible of the portion of bow to be used. However, this is the first work we are aware of that has used the resources of a computer to help the player handle the substantial work involved. We hope to develop a program that will make the task of exploring a variety of potential bowing strategies much easier, thus allowing the player to become more experienced and successful at exploiting the bow. Eventually, this will result in an improved sound as the player is freed to evaluate rather than calculate.

This paper describes an experiment in providing that help. We took the first two movements of the first solo part for Bach's double violin concerto, which had partial bowing information indicated by the editor, and simulated the size of each stroke as the piece was played. The initial goal was to determine whether the bowings provided were practical, or whether the unwary sightreader would be forced to rewrite some of them, thus rejecting the bowing provided.

Our goal was to inform the musician when a bowing was rejected, and to categorise each stroke according to its ending position. We then added a function that detected when successive measures started with an upbow, and another which notified the player if a bowing proposed by the player required the bow to be in a place far from the position calculated by the simulation. We also produced a 'movie', which would represent the view the performer might get when looking into a mirror while playing (a very useful thing to do in learning how to use the bow).

2. The algorithm

The basic ideas are simple. First, we propagate global and local information to each note. For example, a sequence of notes may appear in a section marked 'piano', so we ensure that each note in that section is marked with a *p*. A bowing may be indicated, so we propagate the effect of that bowing to all of the notes after that mark, until the next bowing is indicated. The result of these passes is that each note now contains information about the context in which it appears to the musician at the time the music is being played.

Then, we calculate the amount of bow to be used on each note. The bow is represented as a fraction between 0 and 1. As the bow moves towards the tip on a downbow, its value shrinks; as the bow moves towards the frog on an upbow, its value increases. If the bow position ever becomes negative or greater than 1, then the bowing used is impractical.

A variety of variables affect the sound produced by the bow. For example, the distance from the bridge affects the intensity of the sound, as do the weight or pressure brought to bear by the right hand and arm. We have selected the amount of bow used as the only parameter which will be simulated. The reason for this is that we have found this parameter to be critically important in the development of a string player.

One of the advantages of this simulation is that its calculations demand the use of lots of bow, but there is no reason that it can't be modified to take advantage of other information, such as distance from the bridge and weight.

For the sake of simplicity, we assume that traditional time signatures are used by the pieces we analyse. We assign a beat a set proportion of the bow, called the beat's `unitLength`; this is the fraction of the bow's length consumed by a note whose direction is one unit of the time signature. The simulation adds the duration of the note sequence being played to the bow position if the player is on an upbow, and subtracts it if the player is on a downbow. Dynamics are also fractions. A 'piano' is a fraction of the 1.0 allocated to a 'mezzo piano' or 'mezzo forte' (for simplicity, we will think of these as being the same). The value allocated to 'forte' is more than 1.0. To take account of dynamics, we weight the bow length consumed by a note by multiplying it by the factor given by the dynamics. The fact that 1.0 is allocated to the 'mezzo' value means that the average note is expected to be this loud (it has nothing to do with the representation of the bow varying from 0 to 1).

We want to be able to define a speed at which the simulation takes place. One of the inputs to the simulation is a metronome speed which is divided into 60. Again, to take bow speed into account, we multiply it by the bow length for a given note. If the metronome speed is slower than 60, then the duration of the note(s) played will be longer, simulating a slower speed.

3. The implementation language, Haskell

The simulation was written in Haskell, a pure functional language that supports the required symbolic processing very nicely. We provide some details about the implementation so that our algorithm is as clear as possible.

3.1 Values in Haskell

A value may have a ground type, such as integer or Boolean, or it may be a constructor. The function `(:)` is a list constructor; we may write the list containing 1, 2, and 3 as `1:2:3:[]`. Often it is easier to read the syntax `[1,2,3]`, which stands for the same thing.

3.2 Type definitions

A type in Haskell can be defined in two ways. A variant record type is defined using the syntax:

```
data typename = con1 t1 t2 ... tk1
               | con1 t1 t2 ... tk2 l...
               | con1 t1 t2 ... tkm
```

where `typename` is the name of the type, `ti` a type variable, and `con` a constructor. For example, in Haskell, a type that represents a list of integers could be written as

```
data IntList = NilInt | ConsInt Int IntList
```

It is also possible to give types names. For example,

```
type IntPairList = [(Int, Int)]
```

gives the name `IntPairList` to a list type where all of the elements are pairs of integers.

3.3 Function definitions

In Haskell, a function definition is written as a function name, followed by a series of formal parameters,

followed by an = sign, and finally the function body. For example, the function

```
selectSecond a b = b
```

returns the second of its two arguments.

3.4 Patterns

A formal parameter may be a constructor applied to one or more component formal parameters. For example, the function

```
fst (a,b) = a
```

selects the first component of its formal parameter, which is a tuple. Some types have several constructors, and a value of this type is an application of one of them. Functions are written so that they can handle any possibility by giving a function body for each constructor. For example, we would write the function `not` as

```
not True      = False
not False     = True
```

At runtime each line is matched from top to bottom and left to right. Variables act as 'wild cards', matching any constructor application. We could also have written this function using a case function:

```
not x = case x of
  True  -> False
  False -> True
```

The `case` function evaluates its argument, selects the first alternative that matches the argument's value, and returns the value that appears after the corresponding arrow.

3.5 Type signatures

Functions are annotated with their types; these are called 'type signatures'. For example, `fst` is given the type signature `fst :: (a,b) -> a`, meaning that the argument of `fst` is a pair of values of unknown type, and the result has the type of the first component. There are as many arrows as there are arguments in the function definition. For example, the type of `not` is `Bool -> Bool`.

4. The implementation

We used a very simple model to describe music, which will be developed further in later work.

4.1 Types and constants

We defined the bow position, metronome time, location within the music, and note duration as fractions (in Haskell, this is a `Ratio Integer`). Notes that are bowed together (e.g. slurred notes) are given separate durations that are collected in a list. Thus each element of the music stream is either a note sequence or a rest sequence, represented by the following type:

```
type NoteOrRestSeq = ([Duration], NoteOrRestSeqType)

data NoteOrRestSeqType
  = Rest | Note Bowing Dynamics Context
```

A bowing could be an up bow, down bow, or a continuation of the bowing already established by an

annotation of an earlier note:

```
data Bowing = Up | Dn | C
```

The dynamics for a given note could be piano, mezzo (forte, piano), or forte; it could also be the dynamics established by a previous note.

```
data Dynamics = P | M | F | Cont
```

The context of a note sequence is the bowing information inserted by the player. The simulation calculates a context for every note sequence from this information. The possible bow positions are the tip, middle and frog. It is also convenient for the player to be able to lengthen or shorten notes in order to make some otherwise infeasible bowings work out:

```
data Context = Tip | Mid | Frog | Long | Short
```

The type of the music input to the program is a list of note or rest sequences:

```
type Music = [NoteOrRestSeq]
```

The simulation keeps track of the bow position, the movie and various global values that depend upon the music being analysed, as well as a count of the number of consecutive bars that have started on an upbow (`OnBeat`). This is done in a large tuple data structure called a 'state':

```
type State = (OnBeat, Time, Time,
              Location, BowPos, Movie)
```

Another global is the `unitLength` value, which does not appear within the state structure.

The simulation returns one of four possible constructors, each of which describes a possible scenario. The simulation may have processed all of the music successfully (`OKSoFar`); it may have discovered that the bowing doesn't work (`DisasterAt`); it may have counted too many bars in succession which start with an upbow (`UpBowOnDownBeat`), or it may have discovered that the user has an unrealistic annotation which cannot be used in the context derived by the simulation (`BowPositionCannotAdjust`):

```
data MaybeBowing
  = OKSoFar State
  | DisasterAt Location State
  | UpBowOnDownBeat Location OnBeat State
  | BowPositionCannotAdjust Location
    BowPos NoteOrRestSeqType State
```

4.2 Functions

Three functions perform the main work of the simulation. The first processes the music, returning if a sequence's annotations plus the current state cause a problem that must be reported without continuing through the rest of the music:

```
measureBow :: Music -> State -> MaybeBowing
measureBow [] state = OKSoFar state
measureBow (noteseq:noteseqs) state
  = case (nextSeq noteseq state) of
    OKSoFar state' -> measureBow noteseqs state'
    otherResult    -> otherResult
```

The second function, `nextSeq`, calculates the new state that results for a given note or rest representation and the previous state. It calls `checkDownOnBeat` to determine whether there is an upbow on a downbeat, calculates the duration of the given note or notes, uses this value to calculate the location within the music and creates the next frame of the movie:

```
nextSeq :: NoteOrRestSeqRep -> State -> MaybeBowling
nextSeq (durations, (Note direction dynamic context))
  (onBeat, metro, inABar, location,
   bowPos, movie)
  = let onBeat' = checkDownOnBeat onBeat
      direction location
      duration' = foldr (+) 0 durations
      location' = (duration' / inABar) + location
      movie' = ...
      bowPos' = nextBow bowPos context
                direction metro
                duration' dynamic
      state' = (onBeat', metro, inABar,
                location', bowPos', movie')
  in
    if bowPosFarAway context bowPos
    then BowPositionCannotAdjust location'
      bowPos
      note state'
    else if bowPos' > 1 || bowPos' < 0
    then DisasterAt location' state'
    else if onBeat' > 2
    then UpBowOnDownBeat location' onBeat' state'
    else OKSoFar state'
nextSeq (durations, Rest) (onBeat, metro, inABar,
                           location, bowPos, movie)
  = OKSoFar (onBeat, metro, inABar,
             ((foldr (+) 0 durations) / inABar)
             + location, 5/10, movie)
```

The `nextBow` function calculates the new bow position. If the context of the note requires that note to start from a given portion of the bow, then it starts with that position. Otherwise, it uses the formula outlined in section 2, adding or subtracting the appropriate value if the context is `Long` or `Short`:

```
nextBow :: BowPos -> Context -> Bowling
          -> Time -> Duration -> Dynamics -> BowPos
nextBow bowPos context direction time duration dynamic
  = if (elem context [Tip, Mid, Frog])
    then nextBow (translateConToBow context)
                  Ct direction time duration dynamic
    else
      let len = unitLength * (60 / time) * duration
          * translateDynamics dynamic
          more = if (context == Long)
                  then 1/10
                  else if (context == Short)
                  then -1/10 else 0
      in case direction of
          Up -> bowPos + len + more
```

Dn -> bowPos - len - more

5. Results of the experiment

When working out bowings, it is easy to focus on the potential disasters first, then see what can be done to improve the piece overall. We found that interacting with the program produced the same general behaviour. However, it was useful to have so much help with the problem of calculating the implications of our bowing annotations. After carefully working out the disasters, we could then make as many passes as were needed to remove smaller problems, such as working too much at the frog or tip. We include a half page each of the first and second movements of the Bach double violin concerto, annotated with our contexts and bowings, and with the results calculated by the program, giving the position at the end of each bow stroke. They are by no means optimal, but they are an improvement on the original bowings implied by the edition we used.

It is unlikely that these annotations would have been found as quickly by practicing normally. The reason is that it is difficult to create new bowings which use as much bow as possible, if you are not already used to doing so. A string player has only habit to help in playing through the new annotations (which is required to work out further annotations), and it is difficult to be aware of all of the small adjustments subconsciously made to get around discomfort in moving from the middle to the frog, etc.

6. Future work

As will be clear from our second example, composers sometimes like to write music that can't be bowed using a single strategy. Bach was likely aware of the constraints placed on the soloist in the introduction of this new subject (example 2, bar 17) and wanted the notes to sound as they do, the first held as long as possible with an intense core to the sound as the bow comes into the bridge to adjust for the lack of bow, and the second moved lightly over the middle of the strings, somewhat closer to the fingerboard. For this reason, as well as the fact that players use the distance from the bridge to vary dynamics, we plan to add a new dimension that models the distance from the bridge, again dividing it into three zones.

It should be possible to specify a set of sequences which are consulted by the program and used to improve various standard bowing patterns. The reason that we did not try this so far is that it is difficult to work out when the music really does have one of these patterns and nothing else need be taken into consideration. In example 1, bar 22, there are four eighth notes that cannot easily be played starting upbow because they are on strings that have an intervening string between them. Knowledge of this kind would have to be abstracted from the music, and, ideally, embedded within MIDI or something similar using annotations.

An attractive possibility is that of using these annotations to drive a physical modeling or sample playback synthesizer so that it produces a more realistic string sound. This would allow us to use an infinite bow to get around many bowing constraints, making the results more true to the music. It seems unlikely that this can be accomplished completely without the interactive help of a human being, but we have not yet tried to automate this process and undoubtedly much can be done without human aid.

7. References

- [Askenfelt 1989] A. Askenfelt, *Measurement of the bowing parameters in violin playing. II: Bow-bridge distance, dynamic range, and limits of bow force*, J. Acoust. Soc. Am. 86(2), 503-516.
- [Flesch 1939] C. Flesch, *The Art of Violin Playing*, Bk. 1, Carl Fischer, Inc. 1939.
- [Schelleng 1973] J. Schelleng, *The bowed string and the player*, J. Acoust. Soc. Am. 53(1), 26-41.

Konzert in d-Moll

für zwei Violinen, Streicher und Basso continuo
BWV 1043

] = Freq
 - = Middle
 < = Tip
 } resc of prog
 Johann Sebastian Bach

example 1

Vivace 3

Viol. II

Measures 1-20 of Example 1.

example 2

Measures 15-24 of Example 2.