Computer Assisted Performance using MIDI-Based Electronic Musical Instruments

James S. Plank

Department of Computer Science
University of Tennessee
Knoxville, TN 37996
plank@cs.utk.edu

February 16, 1999

Technical Report UT-CS-99-416
University of Tennessee

Available via ftp to cs.utk.edu in pub/plank/papers/CS-99-416.ps.Z
Or on the web at http://www.cs.utk.edu/~plank/plank/papers/CS-99-416.html

Computer Assisted Performance using MIDI-Based Electronic Musical Instruments

James S. Plank*

February 16, 1999

Technical Report CS-99-416, University of Tennessee

Abstract

Many people desire to play a musical instrument, but lack either the time or the skill to play it to their satisfaction. This paper details a method for a computer to assist a performer in performing specific pieces of music on MIDI-based instruments. The general concept is not new. However, the specific method and matching algorithms are. This paper provides motivation, overview, historical perspective, and details on the algorithms used to provide computer-assisted musical performance.

1 Motivation

Consider an amateur pianist. Call him Bob. Bob loves piano music, and can play a variety of easy to intermediate-level piano works. However, Bob would like to play a wider variety of pieces, including:

- Pieces that are technically too challenging for his current level of skill.
- Pieces that he does not have the time to learn or practice.

In short, Bob does not have the time or skill to play the complete variety of music that he would like to play.

The method described in this paper attempts to solve Bob's problem. With this process, Bob connects an electronic keyboard to a computer. The computer executes a program that access a file containing the notes of the piece that Bob wants to play. The file also contains additional information (defined later in this document) that helps drive the performance. Bob then plays the piece by pressing keys on the keyboard. The keyboard does *not* play the notes that correspond to the keys Bob presses. Instead, the keyboard translates Bob's key presses and releases into events that are sent to the computer (typically using the MIDI standard). The computer uses these events to "play" the notes of the piece. These may be played by the computer's sound card, or sent to a sound synthesizer, perhaps attached to Bob's keyboard.

At a high level, the way the computer performs this translation is as follows. Each key that Bob presses instructs the computer to play one or more notes of the piece. These notes are held until Bob releases the key. Moreover, if Bob's keyboard can sense how hard he presses the keys, then this information is used to direct how loudly the computer plays the notes.

In other words, Bob "plays" the piece much like he plays a piece on a regular piano. However, he does not have to hit the correct notes of the piece. Additionally, he does not have to play all of the notes, since the computer can play multiple notes in accordance to one key press. Most importantly, though, Bob still controls the performance in terms of dynamics, expression, tempo, etc. Thus, the computer does not perform the piece for Bob — Bob performs the piece using the computer.

*Department of Computer Scienct, University of Tennessee, Knoxville, TN 37996. plank@cs.utk.edu, http://www.cs.utk.edu/~plank. This research and software is not part of the author's research program at the University of Tennessee. It is a project that the author has performed in his spare time. Nonetheless, the project has merit in the area of computer science, and therefore the Department of Computer Science has graciously agreed to publish this technical report, and to allow the author to post related material on the department's FTP site and HTTP servers.

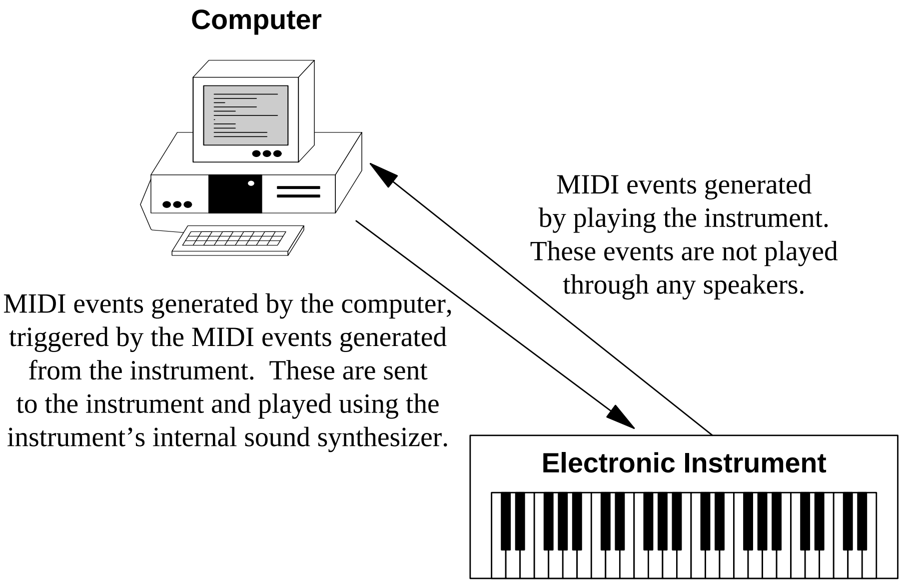


Figure 1: A typical performing environment

In sum, the motivation for this invention is for a computer to assist a performer in playing a piece of music. The performer contols the parts of the performance that make a performance meaningful (dynamics, expression, tempo), but is relieved of the burden of having to learn notes and techniques that are not necessary when a computer is employed. The method attempts to minimize the amount of work (i.e. "practicing") that is necessary for a performer to engage in a satisfying performance of a piece of music.

2 The computing/performing environment

A typical performing environment is depicted in Figure 1. An electronic instrument is played by depressing and releasing keys and pedals, turning wheels, etc. In the description below, I will use the example of an electronic keyboard, but other electronic instruments are possible. The instrument generates musical events such as "note on", "note off", "pedal down", etc. Typically, these are encoded with the MIDI standard, but other encodings are possible. The musical events are *not translated directly into sounds by the instrument.* Most electronic instruments have the ability to "turn local echo off", which means that when a performer plays the instrument, no sounds are emitted through its speakers. This is like playing with the volume turned off. The playing of the instrument simply generates musical events that are sent through an output (MIDI) port of the instrument.

The events go to a computer. The computer has stored a file containing an annotated version of the piece being played. This file contains the notes of the piece and when they are expected to be played. It also contains information about how the events that the performer generates will cause the notes of the piece to be played. The computer matches the input events to the notes of the piece, and emits output events which are to be played by a sound synthesizer. In the picture, the synthesizer is part of the electronic instrument, but it could just as easily be the sound card on the computer.

Thus, instead of having the instrument play notes through a sound synthesizer directly, the instrument provides input to a computer, which plays notes of a specific piece through a sound synthesizer. However, by playing the instrument, the performer has control over the way in which the computer plays these notes. Thus, the instrument, computer, and sound synthesizer combine to become a new instrument whose job is to play the piece of music stored in the input file.

3 The input file

The piece of music being played must first be translated into a format readable by the computer program and stored in a file. This file contains the notes of the piece, plus information on how the input events generated by the performer should play these notes. The input file may be annotated so that the performance can take on any degree of complexity. It can be made very easy, where the entire piece may be played by repeatedly pressing one key on a keyboard, or it can be made very complex, where each event of the piece must be specified by a concommitant event by the performer. In the former case, the piece is easy to play, but gives the performer only limited control of the expression of the piece. In the latter case, the piece is harder to play, but gives the performer more control over the performance.

4 Notation and conventions

In the sections below, the description will assume that the setup in Figure 1 is being used, and that a performer named Bob is playing a velocity sensitive, MIDI-based, electronic keyboard. *Velocity sensitive* means that the keyboard senses how hard a key is pressed, and converts that into a number called the *velocity*. Larger velocities mean that the keys are pressed harder, and if a key press is translated into a note with a large velocity, then a synthesizer normally plays that note louder than one with a small velocity.

MIDI is a format standard for electronic music that is especially well-suited to electronic keyboards. A MIDI-based keyboard emits MIDI events whenever the user performs certain actions. For example, a key press generates the NOTE-ON event, and a key release generates the NOTE-OFF event. These events have the following format:

```
Note-On   key  velocity
Note-Off  key  velocity
```

The KEY field specifies the key pressed. This is an integer value between zero and 127. By convention, middle C is 60, and each half-step on the keyboard equals one unit. Thus, D above middle C is 62, and B below middle C is 59. The velocity is a value between zero and 127. In the NOTE-ON event, it corresponds to how hard the relevant key was pressed. In the NOTE-OFF event, it is typically ignored.

There are other events as part of the MIDI standard. *Control events* are an important class of events. These are events other than NOTE-ON and NOTE-OFF that have an effect on the performace. For example, depressing and releasing a pedal generates two separate control events. A "program change" is another kind of control event that instructs the synthesizer to synthesize sounds from a different instrument. For example, most electronic keyboards can synthesize piano sounds and organ sounds – the PROGRAM-CHANGE event allows the performer to tell the synthesizer which sounds to emit.

Although the KEY field is sufficient to specify the pitch of a note, it is often useful to specify a note by its common musical name. This document employs notation in which notes are specified as *LetterOctave*. Middle C is specified as C0. C an octave higher is specified as C1, and C an octave lower is specified as C-1. B a half-step below middle C is B-1, and D a whole step above middle C is D0. Sharps and flats are specified with '♯' and '♭'. Thus, the note a half-step above middle C is either C♯0 or D♭0. Note that middle C may also be designated as B♯-1.

5 The Method

The method of achieving computer assisted performance is best explained by way of some examples. After detailing three examples, a high-level description of the entire process will be given.

5.1 Example 1: A one-line melody

Suppose that Bob wants to play the first few bars of "Happy Birthday" (Figure 2). We call the computer program that drives his performance "Program A".

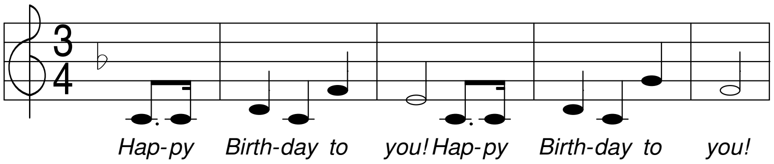


Figure 2: Happy Birthday, melody only.

First, the music is stored in a file that describes the ordering of the notes, as in Figure 3. Note that information concerning the duration of the notes, and where they lie within the piece (e.g. "measure 2, beat 1") is not necessary.

```
C0 C0 D0 C0 F0 E0 C0 C0 D0 C0 G0 F0
```

Figure 3: Happy Birthday, melody only, stored in a file.

Now, to play this piece, Bob simply presses keys on the keyboard. Each key press plays a note of the piece, which is held until the key is released. The first key press plays the first note; the second key press plays the second note, and so on. The identity of the key that is pressed is unimportant in this simple example – Bob may play *any* note to start the piece. However, he knows that the first key pressed plays the first note, the second plays the second and so on.

Since Bob is using a velocity sensitive keyboard, he can dictate the loudness of each note of "Happy Birthday" by how hard he plays the corresponding key. For example, if he wants to stress the note corresponding to the syllable "birth", then he may do so by striking the third key that he plays harder than the others.

What this example shows is a very simple paradigm for letting a performer play a simple piece of music, comprising a single line of notes, by simply playing *any* notes in sequence. The performer controls certain performance parameters, in this case when to play a note, how loud to play it, and when to stop playing it. However, the performer does not need to learn the actual notes of the piece.

Program A's job is rather simple. It first reads the input file and organizes the notes into a linked list. Then it waits for the first NOTE-ON event from the keyboard. Suppose it gets the event (NOTE-ON 65 80) from the keyboard. This corresponds to the performer pressing F0. Upon getting that event, it matches it with the first note on the list (C0), and sends the (NOTE-ON 60 80) event to the keyboard, which then plays C0 at a velocity of 80. Suppose that Bob next releases the key. This generates an event such as (NOTE-OFF 65 64) (note that the velocity is typically ignored). Program A recognizes this as the end of the first note, and sends the event (NOTE-OFF 60 64) to the keyboard, which in turn stops playing C0.

Program A continues in this manner. When it gets a NOTE-ON event, it plays the next note on the list by sending the appropriate NOTE-ON event to the keyboard. When it gets a NOTE-OFF event, it uses the KEY field to determine which NOTE-ON event the NOTE-OFF matches. It then sends NOTE-OFF for that event's key (C0 in the above example).

Note that the NOTE-ON and NOTE-OFF events do not have to come in alternation, and notes may be held as long as the performer wants, not in accordance with how the piece is written.

For example, if Bob's keyboard emits the following sequence of events for playing the first four notes of "Happy Birthday": (i.e. Bob plays the notes F0, F0, G0, A0):

```
(Note-On  65 80)
(Note-Off 65 64)
(Note-On  65 72)
(Note-Off 65 64)
(Note-On  67 85)
(Note-On  69 80)
(Note-Off 69 64)
```

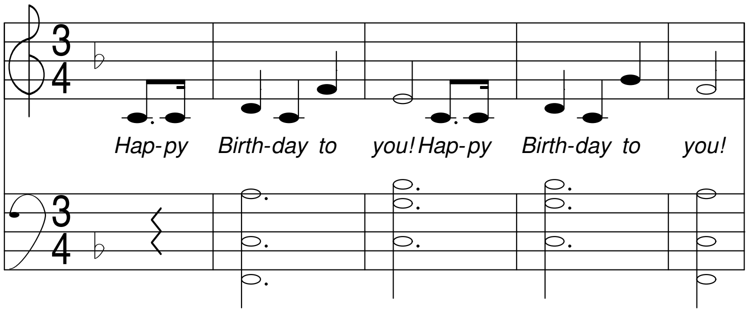


Figure 4: Happy Birthday, melody and accompaniment

```
(Note-Off 67 64)
```

Then Program A causes the following events to be played (C0, C0, D0, C0):

```
(Note-On  60 80)
(Note-Off 60 64)
(Note-On  60 72)
(Note-Off 60 64)
(Note-On  62 85)
(Note-On  60 80)
(Note-Off 60 64)
(Note-Off 62 64)
```

Note that Bob holds the third note past when he releases the fourth note. This is recognized by Program A (which remembers the original identity of the third note played, in this case G0) and the appropriate NOTE-OFF events are generated.

5.2 Example 2: A more complex piece

Now, suppose Bob wants to play a slightly more complex piece of music, like "Happy Birthday" with some accompaniment to the melody (Figure 4).

We'll use a program called "Program B" to help Bob play this piece of music. As before, the piece must be stored in a file, but now there must be some information that relates notes to beats, or at the very least groups together notes that will be played more or less simultaneously. For example, the notes may partitioned into "lines" as in Figure 5, and then translated into a file as depicted in Figure 6.

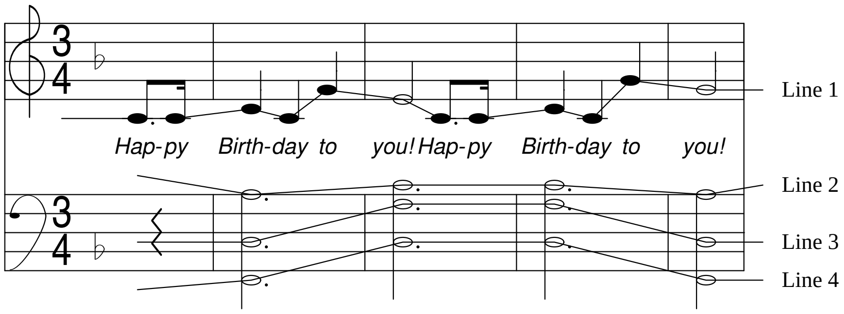


Figure 5: Happy Birthday, partitioned into lines

Program B works in a similar manner to Program A. First, it reads the piece file, and creates a data structure that organizes the notes in time. An example is a linked list with a node for every beat of the piece on which a note is played. We term this a "playable beat." The node is itself a linked list containing every note that is played on that beat. An illustration of the linked list for "Happy Birthday" is in Figure 7.

```
LINE 1
  C0   3/16
  C0   1/16
  D0   1/4
  C0   1/4
  F0   1/4
  E0   1/2
  C0   3/16
  C0   1/16
  D0   1/4
  C0   1/4
  G0   1/4
  F0   1/2

LINE 2
  REST 1/4
  A-1  3/4
  Bb-1 3/4
  Bb-1 3/4
  A-1  1/2

LINE 3
  REST 1/4
  C-1  3/4
  G-1  3/4
  G-1  3/4
  C-1  1/2

LINE 4
  REST 1/4
  F-2  3/4
  C-1  3/4
  C-1  3/4
  F-2  1/2
```

Figure 6: Happy Birthday with accompaniment, partitioned into lines and translated into a file. (**Notation:** Note durations are specified using fractions. For example, 1/4 is a quarter note, 1/8 is an eighth note, 3/16 is a dotted eighth note, etc.)

Now, for the first two notes, Program B works in the same manner as Program A: it waits for the first two NOTE-ON events, matches them to the first two C0's of Line 1, and emits the corresponding NOTE-ON events to be played by the keyboard. At this point, Program B is ready to play four notes (D0, A-1, C-1, F-2), and this is the where Program B differs significantly from Program A. There are several options that Program B has:

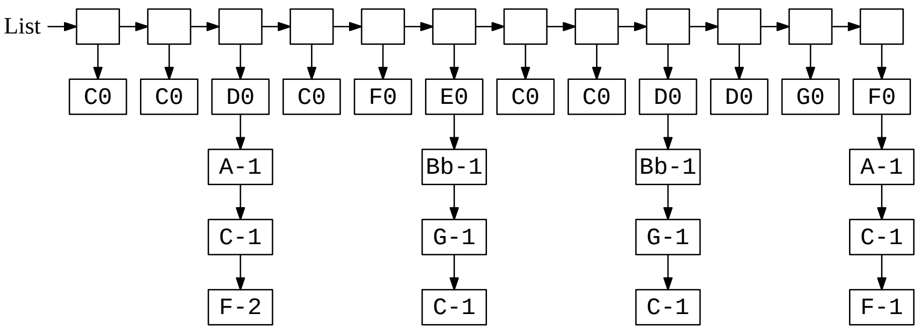


Figure 7: Happy Birthday, melody and accompaniment, converted into a linked list of playable beats.

- When the next NOTE-ON event comes in, generate NOTE-ON events for all four notes, and send them all to the keyboard. When the NOTE-OFF event corresponding to the NOTE-ON event comes in, turn all four notes off. This is an attractive option, but one that requires some careful decisions to be made. We will look at it later in Sections 5.5 and 9. For Program B, we will assume that Bob will press four keys to play the four notes.
- Since the piece specifies that all four notes are to be played at the same time, a reasonable decision for Program B might be to wait for four NOTE-ON events to be generated. When this happens, the events are sorted by their KEY's, and then each event is matched to one of the notes. The event with

the smallest key is assigned to the lowest note (F-2), the event with the next smallest key is assigned to the second lowest note (C-1) and so on. Then NOTE-ON events are generated for each note, each event using the same velocity as its corresponding NOTE-ON event. Each note is turned off when the appropriate NOTE-OFF event is generated by the keyboard. Thus, Bob controls the loudness of each note individually, and how long it is held. For example, he probably will play the D0 louder than the rest, but will hold the other three notes while he plays the C0 and E0 of Line 1.

- Although the above solution is nice, performers often do not play all notes on one beat simultaneously. For example, they may play the highest or lowest note first for emphasis, or they may want to ripple the chords of the accompaniment. Therefore, if Program B is really to deliver Bob the feeling of hearing what he plays when he plays it, it has to generate NOTE-ON events as soon as it receives them — it cannot wait to get all four and then play them. This means that Program B should match each NOTE-ON event to one of the four notes as soon as the event arrives. In other words, it has to guess. This guessing may be done by simple heuristics, such as using information about which keys were matched to previous notes on each line, but in the end, it is just a guess. (The guessing process will be elaborated upon in Section 5.4). Program B then generates the proper NOTE-ON event, and of course, when the corresponding NOTE-OFF event arrives, it turns the note off.

- Finally, a last option is to employ either of the two solutions above, but only to accept NOTE-ON events that are played within a certain time t of each other (for example, 1/10 second). If, t seconds after the first NOTE-ON event is played, no other NOTE-ON events are generated, then the three notes that were not played are skipped, and Program B moves onto the next playable beat. This lets Bob "make a mistake', for example, by only playing three notes on the third playable beat rather than four.

We will assume that Program B implements the last two options: each NOTE-ON event causes Program B to guess which note the event matches, and to emit a NOTE-ON event instantly. However, if too much time elapses after the first NOTE-ON event for a beat, the rest of the notes on that beat are skipped. Program B thus lets Bob play any piece of music by playing a key for every note of the piece. As such, Bob has control over *every note* of the piece: its loudness, when it is played, and when it stops playing. Granted, Bob must have some understanding of how Program B matches the notes he plays to the notes of the piece, but experience has shown that this matches a performer's intuitive notion about which notes be matched to which events.

5.3 Control Events

At this point, it is convenient to talk about control events (such as pedaling and program changes). The simplest thing to do with these events is to pass them straight back to the instrument. For example, if the pedal is depressed, then the computer will get a PEDAL-DOWN event. It should instantly send that event back to the instrument. By passing control events back to the instrument, Bob may employ the pedals in the exact same manner as on a real piano. For example, the damper pedal will allow notes to be held past their NOTE-OFF events, just as on the piano. Performing a program change allows Bob to change the "instrument" while he is playing.

Obviously, other alternatives are possible. For example, a particular control event can be defined to terminate Program B, or skip to the next beat. However, certain control events (such as the pedal or program changes) are best passed back to the instrument.

5.4 Matching Heuristics

When more than one note is to be played on a beat, the notes are sorted by their keys. Ideally, the lowest of these notes should be matched to the lowest note that Bob plays, the second lowest to the second lowest, and so on. However, since the matching must be performed before all the notes are played, Program B must guess.

It is possible to think of many matching heuristics for this guessing. In this section, I will describe the one that I implemented. In practice, I and others have found this to work well. In other words, it is natural for a performer to play in such a way that Program B guesses correctly a large percentage of the time, and therefore that Program B produces music that sounds to Bob like what he means to be playing.

First, the lines are partitioned into two groups: left-hand lines and right-hand lines. The left-hand lines are to be played by the left hand, and the right-hand lines are to be played by the right hand. By convention, I divide the keyboard into two parts — the notes below middle C, and the notes above and including middle C. Any note played that is below middle C is automatically assigned to the left hand, and any note at or above middle C is assigned to the right hand. Note that the notes that the computer plays are unrestricted. The left hand can cause any note on any part of the keyboard to be played. However, the left hand can only press keys below middle C.

```
LINE 1                   TIELINE T1 2 95
  C0   3/16                REST 1/4
  C0   1/16                C-1  3/4
  D0   1/4                 G-1  3/4
  C0   1/4                 G-1  3/4
  F0   1/4                 C-1  1/2
  E0   1/2
  C0   3/16              TIELINE T2 2 100
  C0   1/16                REST 1/4
  D0   1/4                 F-2  3/4
  C0   1/4                 C-1  3/4
  G0   1/4                 C-1  3/4
  F0   1/2                 F-2  1/2

LINE 2
  REST 1/4
  A-1  3/4
  Bb-1 3/4
  Bb-1 3/4
  A-1  1/2
```

Figure 8: Happy Birthday with accompaniment, translated into a file where lines TL1 and TL2 are played with the NOTE-ON and NOTE-OFF events of line LH1

Next, for each line in the input file, Program B remembers the KEY value of the last NOTE-ON event that matched to a note in that line. When Program B has to match a NOTE-ON event to a note, it chooses the line whose last NOTE-ON event most closely matches the current NOTE-ON event. Then, when all the NOTE-ON events for a playable beat have been played, Program B assesses how well it guessed. If it guessed correctly, then it records the KEY values that matched for each line, remembering each for the next matching. However, if it guessed wrong, then it sorts the KEY values of all the NOTE-ON events, and remembers the values that would have been correct for each line. These are then used to perform the next matching.

The actual implementation is more complex than the above description, but the above description captures the basic idea. As stated, experience has shown that this method works well in practice.

5.5 Example 3: Playing multiple notes on one NOTE-ON event

In this section, we will describe Program C, which is more powerful than Program B. What it does is allow multiple notes to be played on single NOTE-ON events. For example, suppose Bob wants to play "Happy Birthday" as shown in Figure 4 in the following way. The notes on the upper bar will be played by his right hand, and the notes on the lower bar will be played by his left hand. Moreover, he would like to be able to play each chord of the left hand with only one note. What Bob does is annotate the input file to show Program C what he wants to play. An example of such an annotation is in Figure 8. This looks much like Figure 6 with the following differences:

- Lines are specified as left-hand (LH) right-hand (RH), or "tie- lines (TL).
- Each tie line is linked to another line, and this link is annotated with a percentage.

Program C reads the file, and creates its linked list like Program B. However, only left- and right-hand notes are in this linked list. The notes in the tie lines are linked to the notes in the lines to which they are tied, and these links are annotated with percentages. The data structure for the file in Figure 8 is pictured in Figure 9.

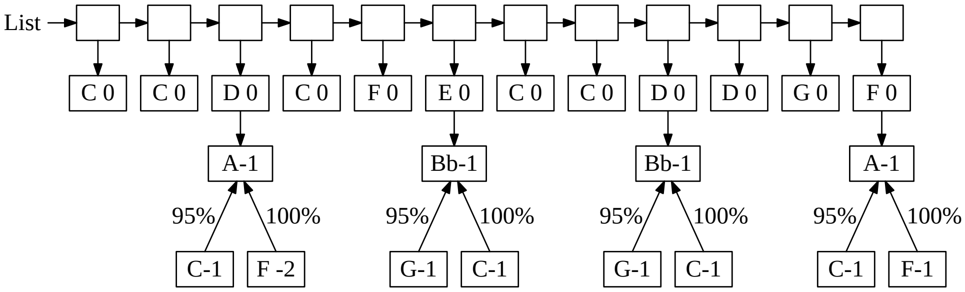


Figure 9: The data structure for the file in Figure 8.

Like Program B, Program C matches NOTE-ON events to notes in the linked list. However, it does not match to notes in the tie lines. If it matches a note that has other notes tied to it, then it emits NOTE-ON events for the note and the tied notes. The velocity of the note itself is equal to the velocity of the NOTE-ON event. The velocity of each tied note is equal to the product of the event's velocity and the tied note's percentage. (Although not shown in the examples, percentages can be greater than 100%). When the NOTE-OFF event comes in for a note with tied notes, then NOTE-OFF events are emitted for all the notes.

In the above example, each tied note has the same starting time and duration as the note to which it is tied. This does not have to be the case. For example, suppose Bob would like to play "Happy Birthday" by simply playing the notes of the melody. One way for Bob to do this is to change the file so that line LH1 becomes "LINE TL3 RH1 85", and to change lines TL1 and TL2 so that they tie to RH1 (with percentages of 81 and 85 respectively) instead of LH1. Then the data structure for the piece looks like Figure 10.

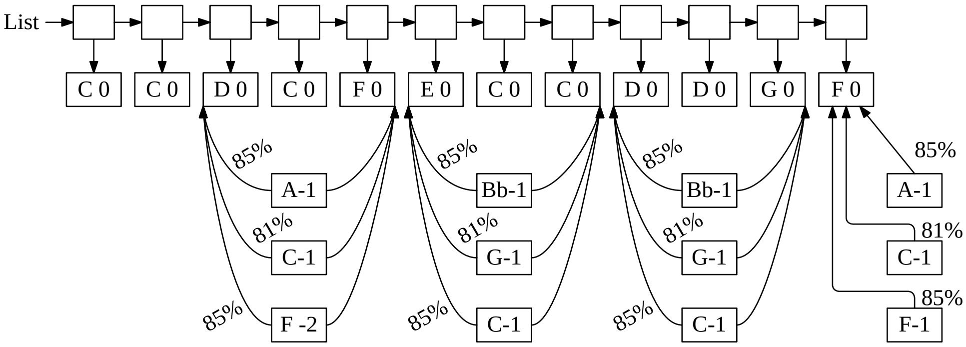


Figure 10: Tying all notes to RH1.

Now, when Bob plays the third note of the piece, four NOTE-ON events are generated. However, when he releases that note, only the NOTE-OFF event for D0 is generated. The NOTE-OFF events for the other three notes are tied to the NOTE-OFF event for the fifth note of the piece.

6 The Process, Restated

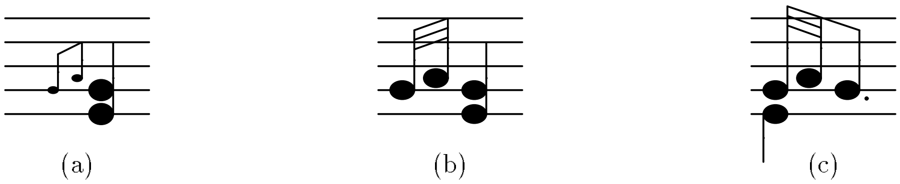
Thus, the process of computer-assisted performance is relatively straightforward. The piece is encoded in a file along with annotations stating the relative order in which notes are expected to be played, and how multiple NOTE-ON and NOTE-OFF events may be tied to single NOTE-ON and NOTE-OFF events. A program then reads this file and converts it into a data structure which orders the notes by when they are expected to be played. It then receives events from the performer, and either passes these events straight through to the synthesizer (in the case of control events), or matches them to the notes of the piece, and emits NOTE-ON and NOTE-OFF events for the matching notes. When multiple events are expected, the program must use some heuristic to perform the matching. A suitable heuristic is one which partitions the keyboard into regions for left- and right-hand notes, and then uses information about past matchings to help choose which notes to match within each partition.

7 Enhancements

The above description describes the general the software process. There are a few enhancements to this process which in practice make the software easier to use, especially with more complex pieces of music.

7.1 Grace Notes

Many classical pieces of music employ *grace notes* as ornamentation. The challenge with incorporating grace notes into the software is their relationship to other notes. If we are only interested in playing one melodic line, then a grace note poses no problems – a program like Program A only cares about the relative ordering of a sequence of notes. However, if one line of notes includes grace notes and another line does not, it is typically up to the performer to decide how those notes are played relative to one another. For example, Figure 11(a), shows a two-note chord with two grace notes ornamenting the topmost note. Figure 11(b) and (c) show two possible ways of playing these grace notes. Either way is perfectly valid, and up to the performer.



(a) (b) (c)

Figure 11: Music with two grace notes, and two valid ways of playing them.

This poses a problem to Programs B and C, because the program cannot assign a playable beat to the grace notes without deciding *a priori* whether to play the grace notes on or before a beat. The solution that I have adopted is to treat grace notes as special notes that may be played at any time before or on the beat that they precede in the music.

7.2 Trills

Trills provide a special challenge to this paradigm because the music does not specify how many notes are actually played during a trill. That is left up to the performer who often doesn't play a set number of notes, but simply trills as fast as he or she can for the required period. Like grace notes, trills are treated as special notes in my software. When the computer receives a NOTE-ON event that it matches a note that is to be trilled, it then waits to receive a NOTE-ON event for any key that is a half or whole step away from the key that triggered the first NOTE-ON event. These two keys are then set to be the trill keys for that note, and as long as these notes are alternated, the trill is continued. However, the software continues to attempt to match other notes, and as soon as the note following the trill in the trill's line is matched, the trill is stopped.

Thus, to perform a trill, the performer simply trills two adjacent keys on the keyboard. An easy way to stop the trill is for the performer to play a different key adjacent to either of the trill keys. This typically matches the next note in the trill's line and thus stops the trill.

7.3 Ripples (rolled chords)

Some pieces of music (an obvious example is Chopin's Etude, Opus 10/11) specify the performer to "roll" a chord instead of playing all notes simultaneously. Ripples necessitate a change in the matching heuristic defined above. Instead of trying to use past information to match the notes of a ripple, the computer instead simply matches the first note played to the lowest note, the second to the second lowest note, and so on. Otherwise, the performer may end up rolling seemingly random notes in the chord.

7.4 Mistakes, hand separation

In Section 5.2, a certain class of mistakes – hitting too few notes when playing a chord – is addressed. It is suggested that if too much time passes between NOTE-ON events, Program B should assume that the performer has omitted some NOTE-ON events, and it should move on to the next beat. There are a few other classes of mistakes that may be addressed as well. First is the case of the performer hitting too many notes on a beat. For example, suppose that four notes are to be played on a chord, and the performer presses five keys. As described above, Program B will play all four notes of the chords, plus a note from the next playable beat. The program may instead be modified to keep track of the tempo between adjacent beats. This is a function of the actual time between when notes from these beats are played, and the duration of the time between the beats in the music (e.g. a quarter note). If a NOTE-ON event occurs within a certain time of the previous beat (for example, 0.1 second), and the previous tempo suggests that the next note should not occur for a much greater period of time (e.g. 0.5 second), then the program can discard the NOTE-ON event (and subsequent NOTE-OFF event) as a mistake.

When lines are partitioned into right and left hands, there may be playable beats (for example, the first beat of Happy Birthday), when the program is expecting to get NOTE-ON events from only one hand. A question is then what should be done when instead the performer plays a note on the other hand? There are three possible solutions to this problem:

1. Ignore the event (and of course the subsequent NOTE-OFF event).
2. Skip the current beat and subsequent beats until there is a beat which expects a note from that hand.
3. Change the data structure so that two lists are maintained – one for the left hand and one for the right hand – and process these lists independently, so that the hands each play in their own tempo. I call this "hand separation".

The solution that I have adopted in my code is a hybrid of these three. First, the music file keeps explicit track of the measures, and measures may be specified as "hands separated" or "hands together" (the default is hands together). If the hands are separated, then within a measure the two hands may play at different tempos. However, if one hand reaches the beginning of the next measure before the other, then both hands skip to the beginning of that measure. If the hands are together, then the tempo estimation (where tempo is a function of the speed at which the two previous beats were played) is used to decide whether the event is ignored or whether beats should be skipped.

For fast pieces where one hand is playing many more notes than the either (e.g. starting at measure 135 in Chopin's third ballade), hand separation makes playing the piece more natural. For slower pieces, or when both hands play in tight synchronization, playing with the hands together works better. In either case, it sometimes happens that the performer makes some mistakes and can't figure out where he is supposed to be in the music. A graphical user interface would help greatly with this problem.

8 Current Status

The current status of this project is as follows. I have defined a format for the piece files, and written some code that lets a user create these files in several stages using a keyboard and metronome. Since the format is an ASCII format, the user may also create files with a standard text editor. The files may be annotated with grace notes, trills, ripples, and the specification of hands together/apart. I have put many classical piano pieces into this format, which may be obtained on the web at:

`http://www.cs.utk.edu/~plank/plank/music/musfiles`

I have written a version of Program C that runs on PC's using the Linux or FreeBSD operating systems. My keyboard is connected to the computer through the joystick port of the computer's sound card, and are accessed using a driver from 4Front Technologies called "Open Sound System" (*http://www.4front-tech.com/linux-x86.html*). The program written in C (approximately 3000 lines), and has a Tcl/Tk front end (136 lines). Besides allowing the performer to play pieces in real-time, Program C creates a MIDI file so that the performer can later play back and post-process what he has played. All the above enhancements (e.g. grace notes, trills, ripples, mistakes, hand separation) have been incorporated.

The program is available on the web at:

`http://www.cs.utk.edu/~plank/plank/music/musplay`

and MIDI files containing some performances created with this program are available at:

`http://www.cs.utk.edu/~plank/plank/music/midifiles`

9 History

As stated in the introduction, the idea behind this method is not new. In May, 1977, U.S. Patent 4,022,097 entitled "Computer-aided musical apparatus and method" was issued to Christopher E. Strangio. In the patent, Strangio describes a machine where sequences of notes may be stored in one of many memory banks. A keyboard is then partitioned into regions, and whenever a key from a certain region is pressed, the next note from that region's memory bank is played. This patent was the inspiration behind keyboards from Casio where special keys triggered the playing of sequences of stored notes. Strangio's patent has expired.

Strangio's patent inspired some other pieces of work, such as the "radio drum" by Max Matthews from Stanford University, where one conducts musing a special pad that senses where a conducting wand is, and adjusts the tempo and volume of a piece of music accordingly. Additionally, Stephen Malinowski implemented Strangio's idea on MIDI-based keyboards as part of his Music Animation Machine [1]

Finally, a piece of software called "Instant Pleasure" was written and marketed for Macintosh computers, which targeted Strangio's patent for MIDI-based keyboards. I do not have any further information on "Instant Pleasure."

One difference between these pieces of work and the method described above is the playing of multiple notes on a beat. Strangio's patent deals with ths problem by partitioning the keyboard into regions, and each region is responsible for a specific collection of notes. Evidently his machine had two such regions, one for the left hand and one for the right hand. Thus, the performer could strike two keys on a beat – one with the left hand, and one with the right hand. If multiple notes were to be performed on a beat, they had to be triggered either by the key from the left hand or the key from the right hand. Malinowski's software is a little more limited – only one key is pressed for each beat. If multiple notes are to be played on a beat, they must all be triggered by one key press. As depicted in Malinowski's video (see his web page for a description), he has developed some very creative techniques for playing fast pieces with his software.

The method in my software allows the performer to specify an arbitrary number of key presses per beat. They keyboard is partitioned into two regions – one for the left hand and one for the right hand. Within each region, the input file can specify that any number of notes be played per beat. The matching of key to note is performed with the heuristic of section 5.4. The intent of this feature is to give the performer more control – chords feel like chords, and multiple melodic lines can be played by a single hand, which has explicit control over each line.

[1] Please see `http://www.well.com/user/smalin/` for a description of the Music Animation Machine.

10 Acknowledgments

The author thanks Don Pederson and Henri Casanova for their discussions on this research area, and Stephen Malinowski for encouragement and providing much needed historical perspective.