New Techniques for Enhanced Quality of Computer Accompaniment

Roger B. Dannenberg
Computer Science Department and
Center for Art and Technology
Carnegie Mellon University
Pittsburgh, PA 15213 USA

Hirofumi Mikuin
Yamaha Music Technologies, USA Inc.
Wood Island Building, Suite 2B
80 East Sir Francis Drake Blvd.
Larkspur Landing, CA 94939 USA

Abstract

A computer accompaniment system has been extended with several techniques to solve problems associated with reliably recognizing and following a real-time performance. One technique is the use of an additional matcher to consider alternatives when the performance could be interpreted in two ways. Another is to delay corrective action when data from the matcher is suspect. Methods for handling grace notes, trills, and glissandi are also presented. The implementation of these and other techniques to produce an enhanced computer accompaniment system is discussed.

1. Introduction

The basic technology for real-time computer accompaniment of live musicians was first presented in 1984 (1,2), however the first systems left a lot of room for improvement. We have been developing new computer accompaniment systems for two years in an effort to obtain higher reliability in the accompanist and also to deal with a more complete musical vocabulary. Like the first systems, the task is to accept a score describing music to be performed by a human and machine. The machine listens to the human performance and synchronizes to it in real time.

A variety of techniques have been developed to increase accompaniment reliability. An important improvement has been the use of a limited amount of non-determinism or parallelism to allow the system to follow two competing hypotheses until one is seen to be superior. A second technique is to delay output from the matcher in order to create a short time window during which decisions can be reversed. Even traditional scores are not completely specified and we have incorporated several extensions to our matcher to deal with grace notes, trills, and glissandi. We indicate the ornamentation in the score so that the system can have an expectation before the ornament is performed. The input is processed according to expectations.

In this paper, we will describe the structure and implementation details of an advanced polyphonic computer accompaniment system. To begin with, in Section 2, we will show the reader the basic structure of this accompaniment system and provide some terminology. In Section 3, we describe how we implement non-deterministic matching. Then in Section 4, we show how ornamentations such as trills or glissandi are handled. In Section 5, we explain the idea of delayed decision making. Finally, in Section 6, we will describe miscellaneous improvements we have done.

2. Basic Structure

In this section, we present the basic structure of our accompaniment system. We will describe only the funda-
mental ideas of accomplishment systems here. For more
details, see the previous lectures (1, 2).

For convenience, we will first define some terms. Be-
fore using a performance, a score is read into the sys-
tem. The score has two parts, the solo score which is to
be played by the human and the accompaniment score
which is to be played by the system. To distinguish what
the computer wrote (the solo score) even when it actu-
ally played, we will call the latter the performance or
simply performance. Time in the score is called virtual
time to acknowledge the difference between the notated
time and actual or real time. The score consists of a set
of events in time. In our work, the solo score consists of
note-on events corresponding to pressing a key on a
keyboard or depressjng a pitch from an acoustic instru-
ment. A compound event is a group of events, which
are played at the nearly same time. Since our events are
note-beginning, a chord is the most common compound
event.

Our accomplishments system has three impor-
tant parts.

Preprocessor This module processes input from MIDS:
IN as makes compound events into timing infor-
mation. If this preprocessor gets two or more
events within a short time period, it will put them
together and make one compound event. Also, it
detects and processes only compound events.

Matcher The matcher compares the performance to a
stored solo score. The Matcher reports correspond-
ences between pitch performance and the solo score
to the Accomplishment part. To finish the matching
computation within reasonable amount of time, it
uses only a small portion of score at any given time.

We call this section of the score a window.

Accomplishment The accomplishment module plays the ac-
complishment part given in the score. It changes its po-
tions and range in real time based on information
from the matcher.

Because we use MIDI for input and output, commer-
cial keyboard controllers and pitch detectors can be used
for input, and synthesizers can be used for output. Al-
though necessary for performance, we do not consider
this to be part of the system.

3. Multiple Matchers?

Motivation The matcher considers only a subset of
the score called the window, at any given moment. If
the solo player is not playing inside of the window, the
matcher will not be able to make note-tomatch. One
problem can be reduced by making the window larger,
but since computation is proportional to the window size,
the problem cannot be eliminated. One particular prob-
lem arises when the window temporarily stops playing.
The accomplishments continues independently using the
soloist's stored information. Where should we expect the
soloist to start playing? Illusory gestures are that the
soloist will either remain where he stopped or will return
in synchrony with the accomplishments. Unfortunately,
this may not be feasible for the window to again both of
these locations. Our solution is to allow multiple match-
ers operationally placed at places likely to match.

Matchers as Objects. We first consider the matcher as an algorithm, which finds a match between
the performance and the stored score. We have changed
our point of view and implementation from procedure to
object so that we can create as many matcher instances as
we want. By this change, we can create a new matcher
whenever we need it, and we can dispose of it whenever
we do not need it.

Creating Matchers. The matcher is a time consuming
module, so we do not want to have many matchers
running all the time. Only when we are particularly un-
certain about the soloist's position will we invoke another
matcher and compute its window on all alternate location.
We will create a new matcher when we have moved from
one recognizable guess about the position of the soloist.
Typically, this happens when the solo player stops his
performance or when he plays many notes within.

In our system, we check the "virtual time" of the accom-
plishment and the "virtual time" range of the matcher's
window. If current "virtual time" is not in that range,
we will create a new matcher at that position whose
"virtual time" is equal to the current accomplishment virtual
time (see Figure 1).

Terminating a matcher. When one of the matchers
finds a match, we terminate the other matchers. We keep
the object in an inactive state until it is needed again.

4. Trills and Glissandi

In our previous systems, all notes were defined in ad-
vance. If we think of a compound event as the event,
we matcher's job is fundamentally to find a one-to-one
mapping between "performance" and "score" (see Fig-

ICMC Proceedings 1988
But this is not true in case of trills or glissandi, because these ornaments depend on the player and are not explicitly specified in the score. For instance, it still does not specify how many notes to play, or does a glissando always specify what pitch to begin from or exactly what notes to play. This is determined by the player and may vary from performance to performance. For example, we cannot rewrite the score from trill(d,a) into [a,b,a,b,a,b,a,b] because we do not know how many iterations there will be. In order to cope with this kind of unpredictability, we need to prepare some mechanism to find a many-to-one mapping (see Figure 3).

Although we could change the matcher, we are hesitant to increase its complexity and decrease its performance. Instead, we use the preprocessor to make one special compound event from the trill glissando event in the score. Because the preprocessor converts trills into single events, the matcher does not need to consider whether the event is special or not.

The Preprocessor. The preprocessor consists of a finite state machine. When it detects special symbols like "trill" or "glissando" in the score, it changes its internal state from "normal" to "trill/glissando". It also gets some other information such as expected termination time of the glissando from the score. The character of these annotations is that notes are far denser than a quarter note. Thus, the preprocessor will not exit the trill/glissando mode until one of the following conditions: the preprocessor gets a long note, the termination time of that ornamentation arrives, or the note value after the trill/glissando is performed near its expected time (see Figure 4).

When the preprocessor enters the glissando state, it sends a special symbol to the matcher. While the preprocessor is in this state, it does not send information to the matcher (see Figure 5).

Informing the Preprocessor. Every time the matcher gets a match, it can request the next note. If this is a special symbol indicating trill or glissando, the matcher tells the preprocessor about it. This causes a state transition in the preprocessor.
5. Delayed Decisions

The machine reports its location to the accompanist whenever a newly performed note leads to a better overall match than any match obtained earlier. Sometimes, this rule does not work. For instance, the soloist may play some grace notes or a trill or a glissando. Sometimes, the preprocessor cannot handle this input correctly, and it sends some extra notes to the machine. In these cases, there is a small possibility of finding a wrong match. To prevent that kind of accidental matching, we should avoid trusting all reports from the machine. The Accompanist can reliably trust the match at the n+1 note if a match of the n-1st note was reported the last time. This consecutive match increases the accompanist's confidence about where it should be looking. But, if a new match is not consecutive to the last match, the system should be suspicious about that match because it indicates that the player made some mistake recently. In such a situation, the machine delays its report to the accompanist. If the soloist is playing grace notes, a trill or a glissando, the next note will come soon. Then the preprocessor will get a note within the delay time, realize something is wrong (because the match cannot match the note) and the delayed report is canceled. If nothing happens to cancel the report, it is sent to the accompanist.

We delay reporting a match only in case of a dubious match, and only for a short period of about 150 ms. Nevertheless, we have to compensate for this delay time. If we suspend a match N ms, and speed is defined as:

\[ speed = \frac{virtual.time}{real.time} \]

delay in virtual time will be:

\[ virtual.time_{delay} = N \times speed \]
(performance) 0010 1010 0001
(scores) 0010 1011 0001
cor.
(result1) 0000 0011 0000
(scores1) 0010 1001 0001
AND
(result2) 0000 0000 0000

Figure 7: bitwise operations.

6.1 Octave Equivalence

We decided to neglect octave differences in pitch. By doing
this, the sitarist can play in any octave. Also, he can
play an opened chord even if it is specified as a closed
d chord. Moreover, there are some drawbacks to dropping
the octave information. Since we treat C3 the same as
C4, we have a small risk of getting a wrong match. Al
though this approach depends upon the music and the
performance, we are happy with this choice, especially
in light of the optimization described next.

6.2 Using Bit Vectors

To deal with polyphonic music, we use bit vectors as
system nets. The vector is a 2 byte integer, whose
LSB corresponds to 'C', and whose next bit corresponds
to 'C♯', and so on. In this data structure, we can in-
clude any combination of notes in 2 bytes. This is pos-
sible because we limit the note range from 0 to 127. If
we used pitch value in the full MIDI range (0 - 127),
we would need 16 bytes. Because matching calculations
are time critical, it is very important to make use data
and operations as compact and fast as possible. By taking
a bitwise 'exclusive or' between the score and perfor-
mance event, we get the result, the bit in the differ-
ence between the two. Then we 'and' result with the
score. We then have 'result1', which shows which notes
in the score we did not played. See figure 7. By the first
operation (exclusive or), we know the difference, which
includes unplayed notes and extra notes. Equal notes
may be in a grace note, so we choose to ignore extra notes.
This is accomplished by the 'and' operation. After that,
we have only unplayed notes. (This can be simplified
slightly: unplayed = result1 & score ) By counting the bit,
we can get the number of unplayed notes. If this number
is smaller than some constant which is associated with
the number of the notes in the score event, we will treat
it as a match. Thus the accompanying system can allow
the operator to play a difficult chord imperfectly, and can
report a match earlier than it would if it waited until a
chord was completed.

Let's take figure 7 as an example. The system of un-
played notes is 1, and the number of notes in the score
event is 4, so 1 unplayed note is allowed. Thus, we say
that these two match.

The merits of these operations are that we can avoid
increasing set operations in the matcher which
would be required if we were to use other data struc-
tures. These operations also effectively eliminate the
twelvetone system, and allow the performer to add some extra
notes without penalty. An interesting consequence is
that notes can be perfectly omitted from score events.
For example, instead of [C E G], one can specify [C] in
the score. Our matching algorithm will evaluate [E G]
automatically from the performance, and report a match.
In the case, any sound with a C, or just the single note
C will match.

7. Evaluation

We succeeded in using a very robust system using the
ideas described above. The techniques for finding grace
notes work especially well in comparison to earlier sys-
tems. A problem arises when the sitar does not play
correctly just before the beginning of a pull or glissando.
In this case, the matcher fails to accept the preprocessor
note text note; the preprocessor then fails to change
its internal state, and it sends all of the ornamenta-
tion notes to the matcher. The matcher will be confused
by the unexpected ways and will not be able to find a match. Even if the subject, the second match, which is located near the current virtual time, will find match when it slips through till or glissando.

8. Future Work
Viewed as an expert system, the computer accompaniment system is still in a knowledge-acquisition phase. We have improved the system performance by causing refer processing models based on our intuitive understanding of how we, as a musician, process and acquire third performance information. In the future, we are planning to improve the system further as outlined below.

As per our, we have to specify wills or glissando by hand. If the player waits to make the score by giving a real-time performance, we need an automatic detection of glissando and pitch. Also, it would be very nice if we could provide a very useful general and unified environment such as we could play, record and edit music easily. For example, the CMD Musician’s Workbench Project will consist of graphic score editors that exchange data through a common music representation used for performance capture and synthesis. The ideal facility is another example of a score manipulation system [5].

Performers sometimes ignore expert signs, miss the code, or skip a page of music. In our implementation, the score is in a one-dimensional array. Thus, we cannot express some musical layout structure specified in the score paper such as a "pause" or "trills" etc. By adding structures to the score to encode repeats or branches, we can use an extra machine of discourse when the player jumps to the wrong spot.

Once our system gets completely lost, not only chance for recovery it if the performance starts following the accompaniment. (Usually but we keep the window of one machine around the current virtual time of the accompaniment.) To prevent correctness, we need to have a list machine, that is the capability of searching the whole score when all else fails.

Improvisation is called for in contemporary and music as well as popular music and jazz. The current accompaniment system cannot deal with such variability in the performance; however, small and predictable variations like grace notes are handled very well, and we have used the preprocessor to filter more complex sequences (wills and glissandos). These techniques might we extended to handle improvisatory music. Ultimately, an accompaniment system should "learn" to an unforeseeable rather than it is not. We are investigating entirely new techniques toward this goal [6].

9. Summary
We have extended our earlier polyphonic accompaniment systems in order to handle, glissando, and grace notes. We have also made the system more robust in the face of performance errors. The two primary techniques are the use of multiple matches and the use of delays in reporting matches. Multiple matches allow the accompaniment system to look for matches within disjoint intervals of the score. With just two matches we can make the system much better at recovering from performance errors. By delaying reports of matches in the accompaniment, we can give our grace notes, performance "glitches", and other short-lived mutations. Delaying allow us in avoiding making a hasty decision when the data is suspect.

10. Conclusions
Computer accompaniment offers the composer and performer much of the flexibility that was lost in the transition from live performance to recorded and taped music. Before computer accompaniment becomes commercially acceptable, they must be reliable, inexpensive, and capable of following the full range of standard performance practice. We believe our computer systems have brought us close to this goal.

11. Acknowledgments
The author wish to thank the Yamaha Corporation, the Center for Artificial Intelligence and the Computer Science Department at Carnegie-Mellon University for their support.

References

ICMC Proceedings 1988


