Handout at Acoustical Society of America, May 1990

Real-Time Computer Accompaniment

Roger B. Dannenberg

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
USA

May 1990

1. Introduction

Now that real-time computer music systems are relatively portable and affordable, it has become common to use them in live performance. Unfortunately, most computer music systems offer little more than a tape recorder in terms of their abilities to interact with live musicians. Computer accompaniment was designed to go beyond this ''tape recorder'' model of human-computer interaction [5]. Computer accompaniment is a process in which a computer ''listens'' to a live musician, follows along in a score, and synchronizes an ''accompaniment'' score with the live player. A system has been implemented that can reliably accompany a live musician in spite of tempo changes and wrong notes. Input is acoustic for instruments that produce a single tone (monophonic) and via mechanical sensors for keyboard input. This paper focuses on the pattern matching used to follow the score and on tempo adjustment techniques used to produce a musical accompaniment. Much of this material is taken from a larger article on Music Understanding [6].

In the computer accompaniment task, a computer system is provided with a machine-readable score that contains music to be played by an ensemble consisting of one or more human players. The score also contains music to be performed by the computer. The task is to listen to the live performance by human musicians and play along in synchrony. The task is complicated by the fact that the human performance will vary somewhat from the score in that human performers might vary tempo and articulation according to acoustical conditions and the response from an audience. Also, both humans and pitch detectors make mistakes.

Accompaniment systems can be classified according to their inputs. A *monophonic* accompaniment system follows an input performance that consists of a sequence of single non-simultaneous notes. A *polyphonic* accompaniment system follows an input performance that consists of a sequence of single or compound (simultaneous) notes. (Simultaneous notes are usually called *chords*.) Polyphonic input is difficult to handle in that the order in which simultaneous notes are detected may vary from one performance to the next, and it can be ambiguous whether a group of notes that are detected in rapid succession are intended to be a chord or a sequence of individual notes.

2. Computer Accompaniment

As described, the goal of computer accompaniment is to follow a live performance and synchronize a computer performance according to a precomposed score. A computer accompaniment system has been implemented as several cooperating subtasks that handle various aspects of accompaniment. Information flow is generally in one direction from each subtask to the next, and the separation into subtasks greatly simplifies the system design. The tasks are *input processing*, *matching*, *accompaniment performance*, and *music synthesis* (see figure 2-1). These will be described in sequence.

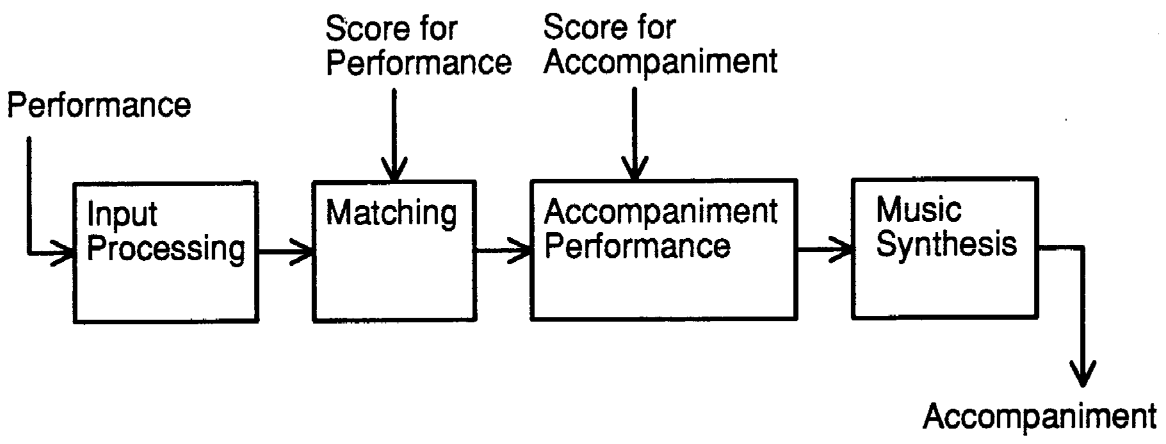

Figure 2-1: Block diagram of a computer accompaniment system.

3. Input Processing

The input processing subtask takes the human performance as its input. Various transformations are performed on this input to produce an abstract symbolic representation of the input for use by the matching subtask. This representation consists of the pitch and onset time of each note.

If the input is audio, then the first step is to track the pitch of the input. At present, this is only possible with monophonic input. The pitch is tracked continuously and quantized to the nearest scale step. Note onsets are assumed to occur when either the nearest scale step changes or when the signal crosses some energy threshold in the positive direction.

Various heuristics can be applied to raw signal processing data in an attempt to increase the reliability of the information. For example, the input processor -- making pitch readings hundreds of times per second -- might not report a pitch change until it obtains three consecutive identical pitch readings. In cases where onset time detection is error-prone, it is sometimes profitable to filter out all consecutive repetitions of a pitch. In effect, a pitch-change detector is substituted for an onset detector. In many cases, the reduction in information is made up for by its reliability.

4. Matching

The *matching* subtask receives input from the input processor and attempts to find a correspondence between the real-time performance and the score. The matcher loads the entire score before the performance begins. Then, as each note is reported by the input processor, the matcher looks for a corresponding note in the score. Whenever a match is found, it is reported to the accompaniment performance subtask. The information needed by the accompaniment performance subtask is just the real-time occurrence of the note performed by the human and the designated time of the note according to the score.

The matcher must meet several criteria. First, it must be a semi-on-line algorithm; that is, it is allowed to look ahead in the score, but it must process performance information as it is received. Second, it must be a real-time algorithm: the processing time allowed for each note must be small and bounded by a constant so that the matcher can keep up with the performance. Third, the matcher must be tolerant of variations in timing as well as wrong notes, extra notes, and omitted notes. Errors will inevitably occur in the performance as well as in the input processor. Fourth, the matcher must have a low rate of false-positive outputs, although the rate of false-negatives can be high. In other words, the matcher should only report matches when there is a very high probability of an actual match.

Since the matcher must be tolerant of timing variations (in fact, the whole purpose of accompaniment is to synchronize in the face of timing variation), matching is performed on sequences of pitches only. This decision makes the matcher completely time-independent. One problem raised by this pitch-only approach is that each pitch is likely to occur many times in a composition. A typical melody might have 170 notes, but only 13 distinct pitches. The most frequent pitch might occur 32 times, or almost one out of every five notes. Whenever this note is played there will be many candidates for a match.

Fortunately, an almost ideal matcher has been developed for computer accompaniment [2]. The matcher is derived from the dynamic programming algorithm for finding the longest common subsequence of two strings. Imagine starting with two strings and eliminating arbitrary characters from each string until the the remaining characters (subsequences) match exactly. If these strings represent the performance and score, respectively, then a common subsequence represents a potential correspondence between performed notes and the score (see Figure 4-1.) If we assume that most of the score will be performed correctly, then the longest possible common subsequence should be close to the "true" correspondence between performance and score. This is merely a heuristic, but it works very well in practice.

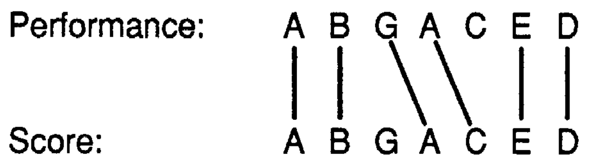

Figure 4-1: The correspondence between a score and a performance.

The longest common subsequence (LCS) algorithm [8] operates by solving the problem on all initial substring pairings of the two input strings. Let P and S be the input strings, and let $P_{<1,c>}$

represent the first r elements of string P. The LCS algorithm constructs a two-dimensional array L in which each element $L_{r,c}$ represents the length of the LCS of $P_{<1,c>}$ and $S_{<1,r>}$. It can be shown (ignoring boundary conditions) that if S_r *does not match* P_c then $L_{r,c} = max(L_{r-1,c}, L_{r,c-1})$ and if S_r *matches* P_c then $L_{r,c} = L_{r-1,c-1} + 1$. Notice that each cell in the array L can be computed in terms of the neighboring cells in the previous row and column. This allows the algorithm to run in time proportional to the product of the lengths of P and S. Figure 4-2 illustrates the array constructed from two strings.

Score: \ Performance:	A	B	G	A	C	E	D
A	1	1	1	1	1	1	1
B	1	2	2	2	2	2	2
C	1	2	2	2	3	3	3
G	1	2	3	3	3	3	3
A	1	2	3	4	4	4	4
E	1	2	3	4	4	5	5
D	1	2	3	4	4	5	6

Figure 4-2: Computing the length of the longest common substring.

The LCS algorithm is not on-line; the longest common subsequence is computed by working backward from the lower right cell of the completed array to construct one or more longest common substrings. The first modification for computer accompaniment is to change the nature of the output. What we want to know is not necessarily the full LCS, but to which element of S (the score) does each element of P (the performance) correspond, if any. A good guess is sufficient, since the optimal guess depends upon the future of the performance. A good heuristic is to report the element of S whenever a new value of L is computed which is larger than any previously computed values of L. (Computation order is important here: the array is computed one column at a time and columns are computed from the first row to the last.) Figure 4-3 has underlined the cells corresponding to matches that would be reported by this heuristic.

Score: \ Performance:	A	B	G	A	C	E	D
A	$\underline{1}$	1	1	1	1	1	1
B	1	$\underline{2}$	2	2	2	2	2
C	1	2	2	2	3	3	3
G	1	2	$\underline{3}$	3	3	3	3
A	1	2	3	$\underline{4}$	4	4	4
E	1	2	3	4	4	$\underline{5}$	5
D	1	2	3	4	4	5	$\underline{6}$

Figure 4-3: In the on-line version, matches are reported when a new maximum is found as shown by underlining.

The modified LCS algorithm is now on-line, but not real-time, because one entire column of L must be computed for each input symbol of P. The height of the column is the size of the score. To avoid a potentially long computation, only a portion of L, called a window, is computed. The window is centered around the row of the expected corresponding score note. Even though most of the array is not computed, identical results will be obtained as long as the ultimate LCS is contained within the window. Figure 4-4 illustrates the use of windows. The window is centered around the last match and advanced by one row per column until the next match is found. For the sake of illustration, a very small window is used; typical window sizes range from 10 to 40 notes.

Performance: **A B G A C E D**

Score:

$$\begin{array}{c|ccccccc}
 & A & B & G & A & C & E & D \\
\hline
A & \underline{1} & 1 & & & & & \\
B & 1 & \underline{2} & 2 & & & & \\
C & 1 & 2 & 2 & & & & \\
G & & & \underline{3} & 3 & & & \\
A & & & & \underline{4} & 4 & 4 & 4 \\
E & & & & 4 & 4 & \underline{5} & 5 \\
D & & & & & 4 & 5 & \underline{6}
\end{array}$$

Figure 4-4: In the on-line version, matches are reported when a new maximum is found as shown by underlining.

One more modification is necessary for use in computer accompaniment. With the algorithm as presented so far, a strange behavior can take place when a wrong note is played. If the wrong note matches some future note in the score but still within the window, then the note will be reported as matching by the heuristic discussed above. Without knowledge of the future, it is impossible to say whether the performer has played a wrong note or skipped ahead in the score. In practice, it is rare to skip ahead, so the heuristic makes many mistakes. This is particularly problematic because if one looks far enough ahead, one is almost sure to find a match. In the LCS algorithm, these ''false'' matches are not a problem because eventually a better match is found, and the ''false'' match is disregarded. It is for this reason that LCS is not an on-line algorithm. For computer accompaniment, however, this problem must be solved differently in order to retain the on-line property.

The solution lies in the recognition that the LCS algorithm is a particular instance of a more general algorithm for finding a string that optimizes an evaluation function. In LCS, the evaluation function is the length of the common substring, but for computer accompaniment, we can modify the function to compute length minus the number of skipped score notes. The new equations for L are: if S_r *does not match* P_c then $L_{r,c} = max(L_{r-1,c} - 1, L_{r,c-1})$ and if S_r *matches* P_c then $L_{r,c} = L_{r-1,c-1} + 1$ Other functions have been considered, but this one seems to work best in practice.

The matcher is an interesting combination of algorithm design, use of heuristics, and outright ad-hoc decisions. Much of the challenge in designing the matcher was to model the matching problem in such a way that good results could be obtained efficiently. This involved compromises in both the musical objective and the formalism.

5. Polyphonic Matchers

Polyphonic matchers have also been explored. One approach is to group individual notes that occur approximately simultaneously into structures called *compound events*. A single isolated note is considered to be a degenerate form of compound event. Now by modifying the definition of ''matches'', the monophonic matcher can be used to find a correspondence between two sequences of compound events. Another approach processes each incoming performance event as it occurs with no regard to its timing relationship to other performed notes. It is important in this case to allow notes within a chord (compound event) in the score to arrive in any order. (Note that the LCS algorithm disallows reordering.) To allow partial reordering, each row of the array L represents a compound score event, while each column represents a single performed note. The resulting algorithm is time-independent. This study was performed with Joshua Bloch and the reader is referred to our paper [1] for further details.

The matcher performs a fairly low-level recognition task where efficiency is important and relatively little knowledge is required. When matches are found, they are reported to the accompaniment performance subtask, which uses knowledge about musical performance to control a synthesizer.

6. Accompaniment Performance

At first, one might think that accompaniment is a simple task once the matcher is present to provide the current location in the score. In fact, things are not so simple because (1) the matcher may not find or report all matches, (2) accompaniment notes may occur in between matched notes, so accompaniment is not simply a matter of waiting for accompaniment notes to be triggered by the matcher, and (3) the accompaniment must deal with match reports that indicate the accompaniment is not presently playing at the proper score location or at the correct tempo.

In order to deal with the problems listed above, the accompaniment performance subtask operates fairly autonomously from the matcher. In essence, the accompaniment advances through the score at a fixed rate, playing the notes it encounters until it receives a message from the matcher. When the message arrives, the accompaniment performer will generally be either ahead or behind the human soloist. The accompaniment performer chooses a strategy to handle the discrepancy and carries out the strategy until the next message from the matcher.

Playing the score at a fixed rate does not mean playing the score at a fixed tempo. In fact, the music is not encoded with respect to beats but in absolute time units. Thus, tempo changes can be an integral part of the score and invisible to the accompaniment performer. The accompaniment performer assumes that variations between the score and the actual performance will be such that a locally linear mapping exists from score time to real time. The accompaniment performer must constantly reevaluate the parameters of this mapping to keep the accompaniment in synchrony with the soloist.

The accompaniment performer has two primary concerns. The first is calculating a good value for the accompaniment rate, the ratio of real time to score time. The second is moving to the correct score location in a musically appropriate way. To compute the rate, a history of recent matches is maintained. Recall that the matcher reports the real time and the score time of each match, exactly the data needed to calculate a best fitting linear approximation. The slope of the best fit line is used as the rate. To maintain robustness, the rate is only changed when the data seem reliable, and several heuristics are used to judge reliability.

The second concern of the accompaniment performer is adjustment to errors in score location. For example, if the matcher reports that the current score time should be 0.3s ahead of the current location of the accompaniment, what should be done? One approach is to increase the rate so as to catch up with the soloist. Another is to jump ahead to catch up immediately. The best strategy depends upon the magnitude of the time difference. If the time difference is very small, then the difference can be attributed to performance interpretation and no adjustment is necessary. Small errors are ultimately corrected by the rate adjustment described above. If the error is significant but still small, say less than 1s, a reasonable approach is to perform the accompaniment very rapidly to catch up or to simply stop the accompaniment and allow the soloist to catch up to the accompaniment. This maintains important musical continuity in the accompaniment. Finally, if the error is large, it is unmusical either to run ahead at high speed or to stop for a long time. In this case, the best strategy is to skip forward or backward in the score. These strategies are encoded as a set of condition-action style rules in the accompaniment performance subtask. More elaborate rules are certainly possible, but these give plausible behavior over a wide range of circumstances.

Although all of the subtasks operate in real-time, there can be a noticeable latency caused by the cumulative delay imposed by each task on the flow of information from note onset to accompaniment output. Since the latency is fairly constant, it can be compensated for by adding a constant (representing total system latency) to the score times reported by the matcher. By adjusting this constant, the system can be made to anticipate the soloist by trying to play everything slightly early, or to lag behind the soloist, playing everything slightly late.

Jazz musicians call this playing ''on top of'' or ''behind'' the beat, respectively, and changes of well under 0.1s have a dramatic effect upon the perception of the human performer. This is interesting in itself, but it also shows that a computer accompaniment system is not just a passive extension to the human player. Instead, a computer accompaniment system is an active participant in the music making process; subtle cues are constantly exchanged in both directions between the computer and human components of the ensemble.

7. Synthesis

The accompaniment performance subtask deals with music at the control level. The output of this subtask is a stream of high-level, real-time commands which specify when to start and stop notes. The commands can also specify many details such as loudness, timbre, vibrato, and articulation if desired. The purpose of the synthesis subtask is to produce sounds according to the commands. This can be achieved through digital synthesis or by mechanically controlling an acoustic instrument such as a piano. The widespread use of the MIDI command interface for music [7] greatly simplifies this task, and the synthesis subtask is reduced to the translation of commands into MIDI commands.

8. Enhancements

The accompaniment system described here has been implemented on several machines including microprocessors and personal computers. In one system, score following is used to display music notation and turn pages appropriately in addition to performing an accompaniment. One can imagine computer accompaniment being used to control other computer graphics and lighting systems, selecting organ stops, and even silencing or correcting wrong notes.

Recently, several enhancements have been made to the accompaniment system described above in order to handle a wider range of input and to improve the matcher. The new system can deal with grace notes, trills, and glissandi, musical ornaments that are often not precisely notated and therefore problematic to the matcher. The system also overcomes the most common cases of matcher failure in which the performer plays outside of the window, making it impossible for the matcher to find the proper score location.

A grace note is a short note played before a longer one. Notated grace notes are no problem for the matcher, but grace notes are often improvised in some forms of music. To deal with grace notes, a slight delay is placed between the matcher and the accompaniment performance subtasks. The matcher also remembers enough state to undo computation for the previous input note. If two notes arrive within the delay period, the matcher considers the possibility that the first note was a grace note and should be ignored. If this leads to an equal or better match, then the matcher backs out of the computation for the first note and cancels any reports to the accompaniment performance subtask.

A trill is a a rapid alternation between two notes. The exact number of notes is left up to the performer. However, the matcher has no knowledge of trills so the trill must be expanded into some number of notes which may not match the performance. Guided by our intuition of our own perception, we have modified the system to treat trills as a special kind of note. When the matcher matches the first note of a trill, it sends a message to the input processor requesting that all further notes of the trill be ignored. This effectively treats the trill as a single musical event. This approach will fail if a bad performance causes the matcher to fail to match the first note of the trill. In this case, the basic matching mechanism will generally recover within a few notes after the trill.

A glissando is a rapid run of notes ascending or descending in pitch. On the piano, a glissando is usually played by drawing the hand across the keys so as to strike each one in sequence. A glissando is like a trill in that an indefinite number of notes may be played, and the mechanism for matching glissandi is similar.

An even more sophisticated approach might be to treat a glissando as a function from time into pitch. One can imagine tracking hand position by noting which key is being pressed at each instant of time. One could use this information to synchronize accompaniment events to the exact pace of the glissando. This possibility has not been explored.

Another problem with the matcher is that the soloist must stay within the matcher's window; in other words, the soloist should never make too many mistakes in succession. This is rarely a problem for experienced players, but it can be caused by an unexpected trill or a momentary lapse of memory, and the effect is serious enough to warrant making some improvements.

To make score following more reliable, we have implemented a two-matcher system in which one matcher functions as described above. The second matcher is enabled whenever the time position of the accompaniment is several seconds outside of the first matcher's window. If the matcher were finding matches and reporting them to the accompaniment performer, this situation would not happen. Therefore, if the accompaniment is outside of the window, it means the matcher is not finding matches; in other words, the accompaniment system is lost!

The rationale for the second matcher is simple: it is reasonable to expect the soloist to try to follow the accompaniment just as the accompaniment system tries to follow the soloist. Therefore, it is likely that the soloist will synchronize with the accompaniment performance. We center the window of the second matcher at this point in hopes of finding matches. If matches are found, we disable the first matcher and follow the second.

With the two-matcher system, we can make some strong claims about when the system will function. If the soloist never makes errors that take him or her out of the matcher's window, or if the soloist always synchronizes with the accompaniment after making such errors, then the accompaniment system will never lose track of the soloist more than momentarily. The improvements discussed in this section were implemented by Hal Mukaino working with the author. Further details are available in a recent publication [4].

9. Conclusions

Computer accompaniment systems work remarkably well due to the robust pattern matching techniques and the high information content of each note.

There are still unexplored aspects of accompaniment for future research. One area of interest is the accompaniment of multiple performers. Although this could be viewed as a form of polyphonic accompaniment, it seems more likely that the best approach is to use multiple monophonic matchers to track each performer and then somehow merge the results to form a composite view of the ensemble tempo and location. This approach has not been tested.

Another interesting area is the extension of computer accompaniment techniques to be more flexible. For example, current systems assume a score is completely specified down to each musical event. Current work is aimed at generalizing scores so that they can trigger complex processes, control graphics, and perform other computations. At the same time, a more flexible system might be used to recognize phrases whenever they occur as opposed to following a predetermined score.

Accompaniment of voice has not been seriously attempted using these techniques. Preliminary testing indicates that, due to the relatively unsteady nature of sung pitches, additional voice-specific preprocessing would be advantageous for vocal accompaniment systems. This has not been explored.

The matchers presented could be used to rapidly scan a database of music. The first few notes could be used as an index into a catalog of musical selections to avoid the necessity of recalling a title.

Other problems arise in the consideration of improvised music in which no predetermined sequence is available for matching and following [3]. It still might be useful to match at a higher level, for example matching against a sequence of chords. Machine understanding of improvised music is a relatively unexplored area, and there is much to be studied.

References

[1] Bloch, Joshua J. and Roger B. Dannenberg.
Real-Time Computer Accompaniment of Keyboard Performances.
In *Proceedings of the 1985 International Computer Music Conference*, pages 279-290. Computer Music Association, 1985.

[2] Dannenberg, Roger B.
An On-Line Algorithm for Real-Time Accompaniment.
In *Proceedings of the 1984 International Computer Music Conference*, pages 193-198. Computer Music Association, 1984.

[3] Dannenberg, Roger B. and Mont-Reynaud, Bernard.
Following an Improvisation in Real Time.
In *Proceedings of the 1987 International Computer Music Conference*, pages 241-248. Computer Music Association, San Francisco, 1987.

[4] Dannenberg, Roger B. and Mukaino, Hirofumi.
New Techniques for Enhanced Quality of Computer Accompaniment.
In *Proceedings of the 1988 International Computer Music Conference*, pages 243-249. Computer Music Association, San Francisco, 1988.

[5] Dannenberg, Roger B.
Real-Time Scheduling and Computer Accompaniment.
System Development Foundation Benchmark Series. Current Directions in Computer Music Research.
MIT Press, 1989, pages 225-262.

[6] Dannenberg, Roger B.
Music Understanding.
1987/1988 Computer Science Research Review.
Carnegie Mellon School of Computer Science, Pittsburgh, PA 15213, USA, 1989, pages 19-28.

[7] Loy, Gareth.
Musicians Make a Standard: The MIDI Phenomenon.
Computer Music Journal 9(4):8-26, Winter, 1985.

[8] Sankoff, David and Kruskal, Joseph B., editors.
Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison.
Addison-Wesley, Reading, Mass., 1983.