REAL-TIME MUSICAL ACCOMPANIMENT

BRIDGET BAIRD
The Center for Arts and Technology
Connecticut College
New London CT 06320 USA

Abstract. The artificially intelligent computer performer is a software program that enables a computer to accompany, in real-time, live performers. The computer is, in effect, a participating member of a musical ensemble. It interacts with the musicians by listening to them and it makes musical decisions based on what it hears and what it knows about music. The computer performer uses parallel processing to speed up its response time. The implementation of the system is discussed, the tracking algorithm and the subsequent computer response are described, and then ongoing and future work are outlined. Musical considerations which have influenced the development of the system are also discussed.

1. Background

The relationship of music to artificial intelligence and cognitive science is complex and fascinating. Questions about where music research fits in these disciplines, what approaches will be most fruitful, and what will be the role of the computer have been studied.

Artificial intelligence and music have a long history of interaction. In 1980 two issues of the Computer Music Journal were devoted exclusively to the topic of AI and music. In 1981 Otto Laske published Music and Mind: An Artificial Intelligence Perspective (Laske, 1981). In 1985 the ACM devoted an issue of ACM Computing Surveys to computer music, and in that issue, in his article on "Research in Music and Artificial Intelligence" Curtis Roads advocates applications of AI methodology and strategies to music (Roads, 1885). At several of the AAAI summer meetings there have been workshops on AI and music. Because of the strong influence of AI on computer music research, parallel processing and neural nets have recently played a significant role in computer music research. The Connection Machine at MIT (Vercoe, 1988) is just one notable example. In 1989, two issues of the Computer Music Journal were devoted to neural nets and connectionism.

More recently there has emerged the field of cognitive musicology, which "has as its goal the modeling of musical intelligence in its many forms" and "...[whose] topics of primary concern ... are those of understanding musical and musicological thought and its link to musical action" (Laske, 1988). The role of cognitive science in this field and whether or not music, because it is based in perception, will employ the same methodologies as other branches of cognitive science, is being studied (Agmon, 1990; Laske, 1988). Also being studied are questions about musical intelligence and whether it is different from other intelligences and in particular, its relationship to linguistics. It is interesting to note that in 1952 John Myhill claimed that "musical thinking cannot be wholly accounted for in computational terms" (Kugel, 1990; Myhill, 1952).

The computer has, of course, played an active role in music research. "For the first time in the history of musical research, the computer program provided a medium for formulating theories of musical activity, whereas prior to its emergence only theories of musical artifacts had existed" (Laske, 1988). Computers are being used to test theories in cognitive musicology, to improvise and compose, as wonderfully versatile computer notation tools, and as instruments, tutors, and performers. It is this last role that we wish to examine more closely.

For quite a few years computers have been used by musicians either to generate sounds that are impossible to produce on conventional instruments or even as substitutes for conventional instruments. In these capacities they have also been used in live performances. But generally the computer performer has been like a tape deck, incapable of altering its performance, and thus forcing the other performers to keep pace with it. It is turned on at the beginning of the performance and performs in exactly the same way for each performance. "An essential part of 'real' music is the live element, the indefinable but undeniable interaction between players and audience which makes music exciting" (Puckette, 1991). A computer performer that acts like a tape deck not only has no interaction with the other performers but also prevents those live performers from any spontaneity or interaction with each other or the audience. What is needed is a more intelligent computer performer, capable of interaction and capable of making adjustments in tempo, dynamics, and expression.

There are many kinds of computer performers, with varying degrees of control exercised either directly or indirectly by the other participants (Pressing, 1990). The immediate goal of this research is to produce an intelligent computer performer that is capable of listening to the other performers, and is able to react and interact in a musical manner, but that is not directly controlled by other performers. The model would be a player in a string quartet, who listens and reacts to other players in matters of dynamics and tempo, but who is not a leader in controlling the performance, except within broad bounds as specified by the score. The computer performer would...
play its part or parts from a score and also has available to it the entire score for the piece. Such a performer would be able to change its tempo and dynamics, and even if the other performers made mistakes, it would be able to guess where they were and respond appropriately. Such a performer would not have the ability to improvise. Although improvisation is an intriguing area of research, it is not the focus here. The kind of computer performer we have described could be used both in live performances (a rock concert or a chamber ensemble, for example) and for training purposes (practicing for a concerto). The ultimate goal of this research is to learn more about musical cognition by producing such a computer performer. The main issues for this research are to devise a good tracking algorithm and then to determine an appropriate musical response.

Several researchers have worked on such a performer. At Carnegie Mellon (Dannenberg, 1984; Bloch and Dannenberg, 1985) a computer accompaniment system was developed that follows a single live performer. In order to track where the live performer is in the score, Bloch and Dannenberg formulated an algorithm that uses pitches and assigns costs to produce the best possible matching with the score. At MIT (Vercoe, 1984; Vercoe and Puckette, 1985) researchers worked independently on a similar system. Unlike the Dannenberg system, this tracking algorithm used both pitch and attack time to make matches. Both systems are limited to input from a single live performer. The Carnegie Mellon system was developed on IBM and Amiga microcomputers and the MIT system was originally hosted on a VAX.

The Artificially Intelligent Computer Performer (AICP) was developed on a Macintosh to implement a tracking algorithm that uses whole patterns of notes in its tracking algorithm (Baird et al., 1989a; 1989b; to appear). This tracking algorithm is described more fully below. Baird extended this work to consider input from several live performers and used parallel processing in order to achieve this (Baird, 1991). Baird, Blevins and Zahler reasoned that the way to emulate how live performers track when they are playing is to consider musical patterns as whole entities and not as isolated notes. Humans process notes as musical motives and not as discrete events in a vacuum. When we hear the opening notes of Beethoven's Fifth Symphony

![Fig. 1.](image)

we consider it as a single musical entity. The Baird et al. tracking algorithm more nearly reflects this situation. Both the algorithm and the entire AICP are described in more detail in the rest of this paper.

### 2. Implementation

The artificially intelligent computer performer is a program which runs on a Macintosh computer. All input and output is in digital form. Although there are many intriguing and interesting problems associated with recognizing pitch from analog sources, we decided to consider only digital information, as specified by the MIDI standard. MIDI stands for Musical Instrument Digital Interface, and is a universal standard for the musical world. MIDI encodes pitch, volume and many other musical parameters. All synthesizers, for example, come equipped with MIDI capabilities. A variety of MIDI instruments are available, and there are even microphones which translate voice and ambient sound into MIDI information. Any device that will give MIDI signals to the Macintosh will work with the AICP. The Macintosh receives and sends signals through a MIDI interface, which can be attached to either the modem or printer port. The Macintosh plays its computer part(s) by sending signals through the same MIDI interface. Several MIDI devices may be connected to the computer, for either input or output, or both. During a live performance, MIDI information is transmitted to the Mac through the interface. The information that is needed for the tracking algorithm is pitch and duration. Other MIDI information about the score, such as meter and tempo, is also used in the program.

The Macintosh will process information from a single live source, but in order to effectively handle more than one live performer, the Mac must be equipped with transputers. Transputers are 32-bit RISC microprocessors which contain their own memory and which are placed inside the Macintosh to give it parallel processing capabilities. The transputers communicate synchronously with each other and with the Mac; each transputer has four input/output channels. When a transputer sends a signal it must wait until the signal is received before continuing on to the next instruction. Transputers achieve parallelism both by having many of them connected together and they also simulate parallelism internally via high speed switching. It is possible to set up multiple processes to run simultaneously on a single transputer. Transputers fit on boards that are placed in NuBus slots; communications among transputers are set up with software commands. The Macintosh communicates with the first transputer on each board. Since our boards each hold a maximum of four transputers, we arranged the transputers in a star configuration, with the first transputer on each board communicating both with the Macintosh and with the three other transputers. Although transputers on one board can communicate with transputers on other boards via a cable link, we did not choose to do this because most of our communication is between the Macintosh and the transputers and not between two transputers. The transputer programs are written in Logical Systems C, and the Macintosh host program is written in MPW C. When the program on the Macintosh first begins, it sends boot code to the first transputers on each board and then the Mac sends boot code for the other transputers via the first transputers. All communication between the Mac and the transputers passes through the first transputers on each board.

When we first started working with the transputers we used a commercially available interface (Express) to make communication easier. Because of the relatively large overhead of this system, and because real-time
processing is a crucial component in this system, we eventually scrapped that interface and have been working out our own communications. This has enormously speeded up the communication time but also enormously added to our programming time. We have established our own broadcast and message passing system, loosely modeled on that of Express. It should be noted that debugging with transputers is a more tedious process than on a conventional machine because there is no direct output device.

After the program starts up, the user can specify the musical score to be loaded. These scores must be in standard MIDI format (most musical notation programs will produce this format). Each part in the score corresponds to a single MIDI channel. The present limit is eight channels (purely for convenience and cost). Once the score is loaded, the user has choices about each of the channels (parts). A channel may be designated as a live channel, a channel to be played by the computer, a file channel, or turned off. File channel means that the part will be played from a file. This file may be either one produced in the MIDI format, or it may be a file which has been saved from a previous performance and is in our own format. Performances can be saved for future use or can be specially construed. Since we are considering multiple live inputs and since our playing abilities are limited by two hands and not much expertise, this is an essential feature. It also allows for exact replications of performances, which assists in evaluation of the program and in debugging. The user has additional options: to change the original tempo of the score, to choose a MIDI port, and to turn off the transputers. The user is also able to select the type of performance: a live performance with combinations of live and file instruments, a computer performance with the computer playing any or all parts of the score, or a replay of the last live performance.

Before the performance begins, the program does a pre-performance reading of the score, much the way live musicians might do. Tempo and beat information are noted. Then each live performer (or MIDI channel) is assigned to one or more transputers. These transputers receive the score for that MIDI channel and during the performance are provided with incoming musical data about that player. At least one transputer keeps a window into the score positioned at what it believes is the correct location of that performer. The previous location and the start of new notes govern where this window is placed in the score. If good matches are not obtained, then the window moves forward as new notes are played. If there are enough transputers, an additional transputer is assigned to a single channel, but this transputer continually checks the beginning of the score. It assumes that the player is starting over from the beginning of the piece. Each transputer uses the tracking algorithm to perform a pattern match of the incoming information to the score for that channel, and then sends back to the Macintosh an estimate of the location in the score of that live performer. The Macintosh reconciles the information from (possibly) many transputers and decides on an overall response. Based on its conclusions it plays its own computer part(s). Since the computer must emulate a live performer, speed is of the essence. Evaluations must be made quickly so that acceptable real-time accompaniment is feasible. During a performance the AICP is faced with essentially two tasks: to determine the location in the score of each of the live performers and to determine the tempo and location at which it itself should play. This involves not only reconciling possibly conflicting information from the various live channels in a musically informed manner, but responding with its own part in a musically acceptable manner.

### 3. Tracking Algorithm

This section describes the tracking algorithm of Baird et al. The first step in determining the correct score position is to treat each live performer separately. The incoming data for a single live performer is matched to the score for that instrument. If there are transputers then this matching takes place on them; otherwise the Macintosh performs the tracking algorithm for a single live performer. This tracking algorithm takes place as follows. The most recently heard notes constitute a performance pattern. This performance pattern is matched to several score patterns. The possible score patterns are determined by looking through a "window" into the score. The center of this window is placed at the best guess as to the current correct score position. Score patterns are taken within this window. The size of the window and the size of the patterns (both performer and score) are governed by processing time, although patterns are not allowed to be too large because of musical considerations. A single performance pattern is matched to each of many possible score patterns and a cost is assigned to each match. The minimum cost from all of these is picked, and both the cost and the location in the score of that best match is conveyed to the Macintosh.

The cost of a match of a performance pattern to a score pattern is based on both duration and pitch information for all the notes in the performance pattern, with the most recently heard notes carrying greater weight. Pitch mismatches incur a relatively greater cost than duration ones. In fact, since musicians may be rather inexact about ending times of notes, some of these "gaps" are smoothed over. Rests constitute a type of "note" and, with some slight modifications, are treated as such. One musical problem is to determine when a performer is intentionally playing a rest and when there is merely a gap in the performance. In absolute time it is perfectly possible for there to be a rest in the score that is longer than a performer's pause between two successive notes. The algorithm takes into account this kind of scenario. In fact, there are four types of individual note matches that are considered. Three of these matches anticipate the kinds of mistakes that musicians are likely to make during a performance.

The first kind of note match is a verbatim match. One note in the performance is matched to one note in the score (either of the notes could be a rest). A cost is assigned for incorrect pitch; the greater the difference in pitches the greater the cost, modulo the octaves. A cost is also assigned for differences in duration. As long as two successive performance notes do not have a rest, the algorithm assumes that the end of the first note is at the
beginning of the next one, so the gaps between notes are smoothed over.

The second kind of note match, an amalgamated match, occurs when the performer plays a wrong note and then immediately corrects it to the right note. Two of the performer’s notes are matched to one of the score notes. The pitch of the performer’s second note is compared to the pitch of the score’s note, and the sum of the durations of the performer’s two notes is compared to the duration of the score note. This example is illustrated in Figure 2.

The third type of match, the held through, is when two notes in the score are matched to a single performance note. The performer has missed playing one note and has instead held the previous note through the time of the second note (see Figure 3). The pitch of the performer’s note is compared to the pitch of the first note in the score and the duration of the performer’s note is compared to the sum of the durations of the two score notes.

The last type of match, the rest, occurs when the performer releases a note early, probably in anticipation of the next note or passage (see Figure 4). The gap that is caused by the early release might be long enough to count, in the computer’s view, as a rest. It is not desirable to shorten the length of time the computer considers a rest because there may in fact be places in the score where this length of time should count as a rest. On the other hand, treating this situation as a rest in some cases would cause undue cost to be assigned in the tracking algorithm. Instead, in this situation the duration of the performer’s pause is added to the previous note and this unit is compared to one note in the score.

In order to calculate the entire cost for a match of a performance pattern to a score pattern, the last notes (most recently heard) are treated first. For each note the four types of matches outlined above are considered and possible costs computed. For all but the verbatim match, a small penalty is added to the cost. The algorithm then considers all four possibilities for all of the notes in the pattern, and adds these to the total cost. The algorithm is performed recursively. The computer has an upper bound on acceptable costs so that obviously unprofitable paths are cut off. At the end of one pattern match there is a total least cost. This process is repeated for each score pattern. For each successive score pattern, the accumulating cost is compared with the best cost so far, so that dead ends are eliminated. Finally, the overall minimal cost is selected from all the score patterns.

This part of the program is computationally intensive and since responding in real-time is a major consideration, the parallel processing helps this part of the program. Through experimentation, it was determined that if the tracking algorithm takes more than approximately 5/60 of a second, the degradation in the computer’s response is too great. This constraint governs the size of the window and the size of the patterns. The size of the patterns is also restricted by musical considerations. When the Macintosh is running without transputers, this restriction on processing time effectively limits the performance to one or two performers and limits the window size to about 5 notes wide.

Once the best cost has been determined, this also gives a best beat as specified by the location of that score pattern. Location in the score is calculate by the total number of beats from the beginning of the piece. These two pieces of information, beat and cost, are communicated to the Macintosh. The Macintosh stores this information and also the time (on the computer clock) when the information arrived.

4. Computer Performer Response

Once the program receives beat information from a live channel and also an estimate as to the reliability of that information (cost of the tracking algorithm) it must decide on its response. If there is more than one performer, then conflicting locations may be indicated by the different performers and then the AICP must, in effect, become a conductor, and determine a "true" location and a "true" tempo.
In these considerations, the issues of time and tempo are extremely important. The computer has its own internal clock, which is keeping an absolute time; each transputer keeps an absolute time also, although because their units are different from the Macintosh's and because any communication between the computer and the transputers takes an (albeit) small amount of time, there are minor inconsistencies. Tempo is not an absolute. There is the tempo of the piece, there is the tempo at which the computer believes the piece is being played, i.e. a conductor's tempo, there are the tempos at which the live performers are playing and then there is the tempo at which the computer might be playing (slowing down or speeding up) in order to catch the rest of the players. The two absolutes in the piece are the internal clock of the Macintosh and the beats in the score. Everything else is relative.

In order to decide on one location in the score, all of the beat and cost information, and the computer time at which it was received from the various performers, are put into an array. A linear least squares fit of beat vs. time is performed to determine the tempo of the piece. This least squares fit is weighted according to the most recently heard information and also according to the reliability or cost of each piece of information. This means that moving lines will have a greater say in determining the tempo, which generally is consistent with musical interpretation. Moving parts often have the melody and thus should take the lead in determining tempo, although moving lines may also denote secondary instruments and it is debatable whether they should be allowed to keep the beat. It is also possible to weight the least squares fit according to the instrument. For example, in a string quartet the first violin might be assigned a greater weight and thus be given more say in determining the tempo.

The linear least squares fit will give the computer not only an indication about the correct current beat but will also give the current tempo, which can be found from the slope of the line. Now the computer must decide on the correct response. At this point the computer believes it knows the correct beat and the current tempo. Chances are the computer is not in the exact correct location in the score and is not playing at the current tempo.

One possible response for the computer is to simply move to the correct location and to start playing the correct tempo. The main problem with this approach is that it is not in conformance with how humans make music. There may be situations in which the computer should shift its location in the score, but only when it is very far from the correct location. And in that situation, the computer might have to wait before shifting if it has only just begun a note. In effect, the computer must be slowed down to human speed. If the computer holds a note for an extremely short period of time, the result is jarring and unmusical. But in most situations the computer should not jump to another location in the score but should slow down or speed up to catch the performers. How much it should adjust its tempo in order to catch them depends on several factors.

The first factor is a musical one and is governed by musical experimentation. What is needed is to determine an interval over which the computer catches the live performers. It is a parameter that is set to between one and two seconds. Less than one second sounds artificial, and more appears to be inefficient. The second factor is governed by the kind of piece that is being played. In a Bach chorale, the tempo should never be unduly fast, whereas in a vivace movement, there is more leeway about speed. Thus there is an upper limit established. There seems to be more latitude about a lower limit, however. An incredibly slow tempo essentially amounts to holding notes, and does not sound terribly out of place even in fast movements. These upper and lower limits are set by the pre-performance score consultation and are governed by the meter and tempo of the piece. In fact, as pieces are read in, several parameters, such as these, are set. But even if a tempo is allowed by the upper and lower bounds, it may be inappropriate because the change is too abrupt. Thus the previous tempo also sets limits on the ensuing one.

Although tempo is perhaps the most important consideration, there are other factors to be noted. As the program plays a performance it also checks the dynamics of the other players. Incoming data is checked against the score and if there are large differences in dynamics then the AICP modifies its response by playing softer or louder also.

It is possible that the incoming information is deemed too unreliable to act upon, i.e. the costs are unacceptably high. Then the response of the AICP is to continue playing, while gradually reverting to the tempo specified in the score. In an earlier version of the AICP, if the computer determined that the live player was hopelessly lost (in its view) then it would stop playing for a short while and then jump ahead to a place in the music just after a tonic cadence (as found by the pre-performance score consultation). It would also play quite loudly. This would correspond to giving a cue to start playing at the beginning of a new section. This kind of jump would only work in classical western tonal music, for only in those pieces could the computer identify the key of the piece and then also be able to identify the cadences.

5. Ongoing and Future Work

At present the author is in the midst of working on the reconciliation aspect of this work. The system works extremely well when there is a single live performer. It is difficult to throw off the computer, even when many mistakes are made, and the computer beautifully follows the tempo of the live performer. When there are several live performers, the computer tracks them reasonably well, as long as they are not themselves way out of step with each other, which is about what should be expected. Much more experimentation needs to be done to tune the weights and parameters that are used in the multiple live input case. There is a difficulty in obtaining appropriate test data. If the live performers' parts are given by files, they tend to have no affect on each other, which is not the case for a true live performance. But in executing live performances on several instruments there is a great tendency for the performers to have a great affect on each other and all play at the same tempo, in which
case the system does quite well. The best data comes from two experienced players, when one of them is asked to lead and change tempo, and the other follows suit, albeit slowly. Perhaps the lesson here is that in most ensembles there will tend to be one tempo, and the computer performer need not worry too much about reconciling wildly conflicting information.

At present parallel processing is used in a very straightforward manner, mainly to give the system greater speed. Different processors correspond to different performers, which begins to emulate the parallel nature of human cognition, but as more is known about musical cognition, this whole area could open up much more.

A very promising domain for further research is to examine the tracking algorithm. It would be preferable if the algorithm not only used patterns of notes, but actually used more musical patterns of notes. This would more closely emulate what humans do in a musical ensemble. This gets into the fascinating and difficult area of human cognition, but as more is known about musical cognition, this whole area could open up much more.

At least, the system should be modified so that musical patterns do not cross obvious musical boundaries, such as ends of phrases or sections. There is much interesting and fruitful research still to be accomplished.

Acknowledgments: The author would like to thank the Knowledge Models and Cognitive Systems program of NSF for supporting this research under grant IRI-9010793. She also thanks her early collaborators, Donald Blevins and Noel Zahler, for their continued input.

References


