SCORE FOLLOWING IN OPEN FORM COMPOSITIONS

John Lawter
Department of Computer Science
State University of New York at Buffalo
226 Bell Hall
Buffalo, NY 14260
U.S.A
lawter@cs.buffalo.edu

Barry Moon
Hiller Computer Music Studios
State University of New York at Buffalo
222 Baird Hall
Buffalo, NY 14260
brmoon@acsu.buffalo.edu

Abstract

Open form score following presents challenges which cannot be adequately met by using the objects currently existing in the Max environment. By using an external object to perform the basic following task, the authors found they were able to generate better and more reliable performance.

1. Introduction

Previous research in the area of open form score following was motivated by problems and limitations inherent in following fixed form scores. Many of these problems arose because of the indeterminate nature of both pitch tracking and human performance.

The indeterminacy of pitch tracking is due to many variables, not only in the algorithm itself, but in the input signal strength and quality. The indeterminacy of human performance, on the other hand, is often criticized, because score following algorithms demand consistency. This is a fundamental contradiction of a so-called "interactive" music, and is the main reason for developing an environment in which indeterminateness of human input becomes a positive, rather than a negative, aspect of score following.

This environment can be broken down into two parts; a low-level algorithm which compares the live performance with a pre-recorded electronic score, and a higher-level algorithm which maps the followers output to variables in the signal processing network. In prior implementations, these two parts where integrated into a single Max patch. The impetus for developing an external object to handle low-level calculations came about due to limitations of processing speed in Max. Although Max can provide minimal latency, the control data rate has to be slowed to avoid excessive drain on the signal processing. Encapsulating the functionality of the low-level algorithm into an external object allowed the circumvention of these problems in handling control data, as well as possible improvements in both accuracy and speed.

2. Design & Implementation

The design of the follower is simple and relatively straightforward, to reduce the overall time complexity of its operations and aid in maintainability. It has one major data structure, an array of linked lists referred to as theScore, and two major operations: addToScore(), which adds an element to the array of lists, and followThis(), which outputs an index and a reliability or confidence level for a given pitch input.

The follower's theScore array has a total of 128 list entries, one per MIDI note. Although this may result in wasted space due to the repeated use of some notes and non-use of others, the nature of the array affords faster access to the individual lists than would a dynamically allocated list or tree structure.

Each list holds a series of Noterecords, one record for every occurrence of a note at a given pitch. The Noterecord structure contains a unique identifier, or index, an array of the first five notes before the occurrence, and the number of an event associated with the note, if any. At this time however, the event data for a note occurrence is ignored by

the follower.

The addToScore() operation takes care of assigning the unique identifier, as well as keeping track of the most recent note inputs, referred to as the "neighbors." The method keeps track of how many notes it has processed, and uses this as a note's index. The five neighbors are taken from an array which functions as a queue, storing the notes in a first-in first-out manner. Upon receiving a note to enter into theScore, addToScore() creates a new Noterecord, inserts it into theScore, into the list indexed by the note value, and initializes the index, event and neighbor fields of the record. After this, it removes the oldest note from the queue and inserts the current note.

The followThis() method does the "grunt work" of the follower. It uses a queue similar to that of addToScore(), to track the five most recent notes. When followThis() receives a note input, it finds the list in theScore for the input's MIDI note value, then scans through this list, comparing its current input and the five previous notes with each Noterecord in the list. It keeps track of the confidence returned by the comparison function, and will return the index associated with the Noterecord that receives the highest confidence rating.

This comparison function runs though a Noterecord's stored list of neighbors from least to most recent, comparing each one to the notes stored in followThis()'s queue. After checking all of the neighbors, the function returns the total number of successful matches.

3. Testing

The preferred method for generating electronic scores for all types of score following is to record data coming from a pitch tracker as a performer plays sequentially through a score. This allows for irregularities of pitch tracking to be recorded, and also allows for filtering data coming from the pitch tracker; this filtering is also applied in performance. For example, the composer may want to filter out oscillations between pitches associated with multiphonics and trills. The authors tested the follower using data recorded from pitch tracker output in an electronic score of 255 notes.

The most important test for experimental purposes was measuring the accuracy of the follower when a score is fragmented, as it would be in the performance of an open form composition. The results of one test run are shown in figures 1a and 1b. In both figures, the x-axis, labeled fragmentation, represents the number of notes which are played sequentially from the score before jumping to a random position in the score. The y-axis of figure 1a represents the percentage of the total number of notes registered at each of the five levels of confidence, and the y-axis in figure 1b represents the percentage of correct score indices registered at each of the five confidence levels.

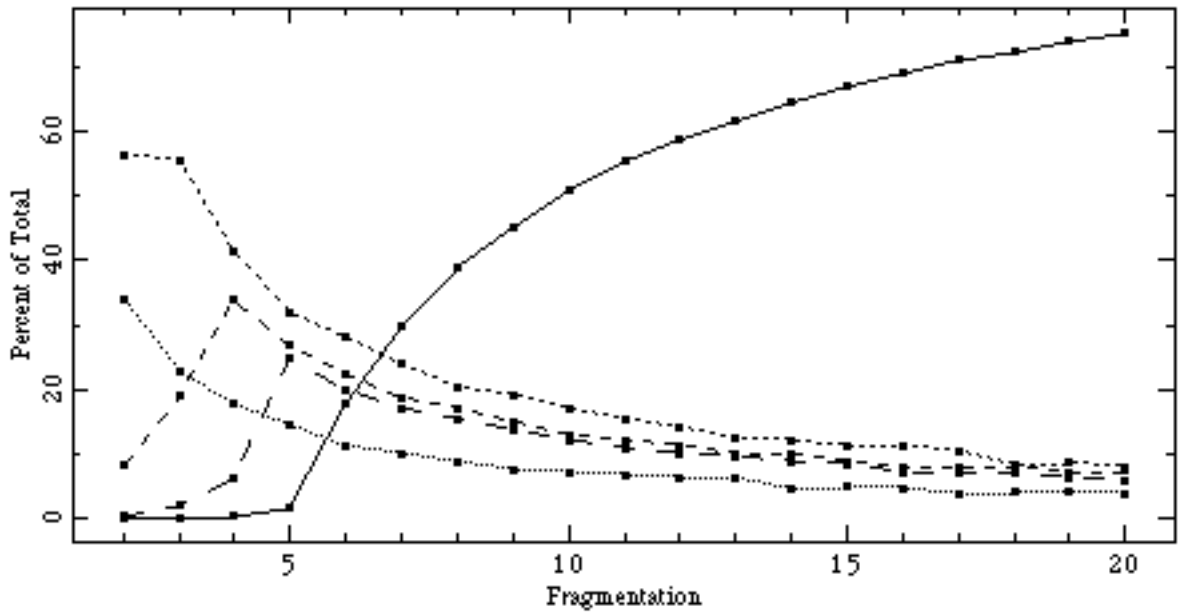


Figure 1a

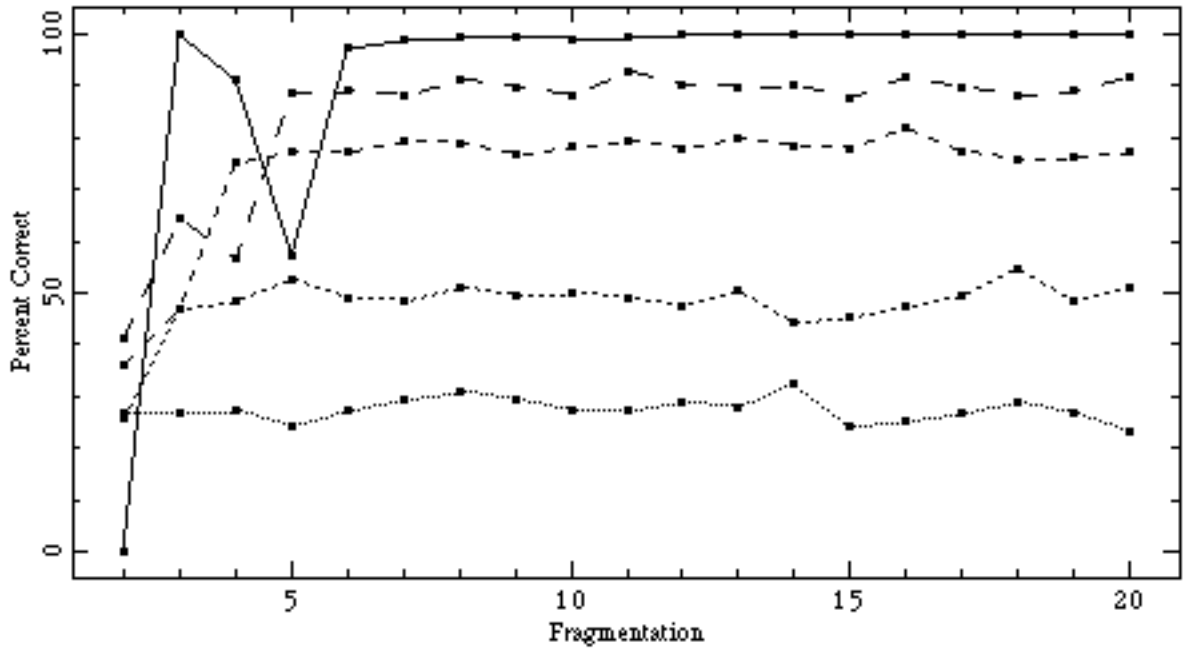
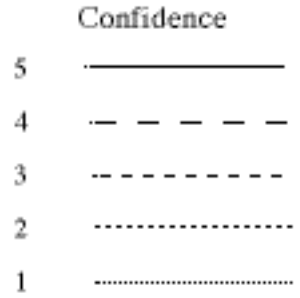


Figure 1b



We can see from these two figures that the follower's reliability decreases if fewer than 10 notes are played in sequence before jumping to a changing position in the score. This effect is due to the size of the "window" set up by the previously played notes. Once the player's position in the score jumps, the data in the window is no longer valid; instead of helping the follower it confuses the comparison algorithm, resulting in lower confidence ratings and inaccurate output. After the performer jumps to another section it will take five notes of error free input for the follower's data structures to be refreshed with usable data.

The follower was also tested for accuracy under several types of errors commonly associated with performance and pitch tracking accuracy. The follower's accuracy was tested with regards to three different types of errors: replacement, underflow and overflow. The replacement type error is caused by a typical performance problem, playing wrong notes. To simulate this form of error, notes performed from the electronic score were replaced by "wrong" notes. The other two types of error, underflow and overflow, are both generally due to fluctuations in the amplitude of the signal sent to the pitch tracker. They represent too few and too many notes coming from the pitch tracker, respectively.

Of these three types of error, overflow poses the greatest problem to the accuracy of the follower output, with replacement a close second. Interestingly, underflow poses few problems for the follower. Underflow causes fewer problems for the follower due to the nature of the comparison algorithm; the presence of excess notes will "push" the useful notes out of the follower's comparison window, thus reducing the total number of successful matches.

4. Applications

The specifications of the follower were determined by the previous implementation of an open form following and processing environment (Moon, 97). These specifications included the output of both a determined position in the score (index), and determined accuracy of this position (confidence) for each note produced by the pitch tracking output. As with all data associated with performance analysis, this follower output can either be mapped directly to signal processing algorithms, or further processed. To date, the most successful processing of follower data has been a kind of low-pass filtration to both remove erroneous data and build up a progressive rating of confidence. This is achieved by rating the index according to the length of a linear progression and multiplying this with an average of the confidence data. The more accurate score index thus produced is used to make global changes in signal processing algorithms analogous to the "events" of traditional score following, except that there is no assumed linearity of events. The rating output is used to make continuous changes in signal processing, mainly at the level of routing and mixing the various signals in the processing network.

5. Conclusion

The follower discussed in this paper was developed out of a need to follow open form compositions utilizing real time signal processing in the MSP environment. Based on experiences in open form score following using only the built-in Max objects, it seemed that better results could be achieved by putting the low-level, "follower" functionality into a single external object, in order to overcome difficulties with speed and the amount of control data that could be processed efficiently.

Although the results of testing the follower are encouraging, there is room for improvement. Some of the test results indicate that the number of "neighbors" used in comparison should be made larger, to increase accuracy in sections with large amounts of repetition. Also, the followThis() method needs to use a "tie-breaking" method, to deal with situations where several Noterecords have identical confidences. At this time, only the first index is returned, which biases the follower in favor of the earlier sections of the score. A second level of comparison in this case should ensure better coverage of the entire score.

The internal data structures may need to be changed. By using linked lists, the follower takes theta(n) time to search through a list of size n. By using a differenct structure, such as a search-tree, this search time could be reduced. As it stands now, this time complexity is not much of a disadvantage, but in following longer scores, or scores in which use large amounts of repetition, it could be useful.

Finally, although the follower is geared towards open form following, it may, with slight modification, be made suitable for traditional linear score following as well. Also, even without modification, the object could be used to track melodic gestures in fully improvised music.

References

Dannenberg, R. 1989. "Real-Time Scheduling and Computer Accompaniment." Current Directions in Computer Music Research. Eds. M.V. Mathews and J.R. Pierce. Cambridge, MA: MIT Press. 223-261.

Moon, B. 1997. "Score following and real-time signal processing strategies in open form compositions." U.S.A./Japan InterCollege Computer Music Festival Proceedings. 11-19.

Puckette, M. and Cort Lippe. 1992. "Score Following in Practice." Proceedings of the 1992 International Computer Music Conference. San Francisco, CA: Computer Music Association. 182-185.

Rowe, R. 1993. Interactive Performance Systems. Cambridge, MA: MIT Press.