A Model of the Process of Listening to Simple Rhythms

David Rosenthal
The Music + Cognition Group
The Media Laboratory
Massachusetts Institute of Technology
20 Ames Street
Cambridge, Massachusetts, 02139
email: dfr@ems.media.mit.edu

Abstract

This paper discusses a computer model of the process of listening to simple rhythms. The model consists of (1) a way of dividing the rhythm into appropriate chunks, (2) a means of constructing *recognizers* for the chunks, and (3) an organization of the recognizers into a hierarchical structure. Each node in the structure is a recognizer for the nodes immediately below it in the hierarchy. As a result of constructing this model, we see that some musical attributes normally considered to be aesthetic are actually necessary for the model's efficient function.

1. Introduction

1.1 *What does the program do?*

The purpose of the program is to express theories about the processes involved in listening to simple melodies.

The input to the program is a series of "beats," which describe what is happening in the melody at successive time intervals. For the purposes of this paper, beats may be thought of as eighth-notes. Thus, for example, a dotted-quarter note is indicated by three beats, the first of which indicates that there is an attack, and the second two of which are "ties," i.e., they indicate that the note continues to sound, but there is no fresh attack. The series of beats is preprocessed into a series of "events," which correspond roughly to notes. An event is a LISP object which contains information about its length (in beats), pitch, whether or not it begins a measure, and pointers to the previous and next events.

The program processes (or, in the parlance I will adopt, listens to) a series of events. It constructs libraries of "recognizers," or LISP objects which can recognize a certain pattern of events. As an event is processed, decisions are made as to whether the event is part of a pattern which the program recognizes, or whether the pattern is a new one, for which a new recognizer should be constructed, or whether the decision should be deferred to a subsequent event. In certain circumstances, the program will construct "higher-level" recognizers (henceforth, hl-recognizers), which recognize patterns of other recognizers - either hl-recognizers or event-pattern recognizers.

The output of the program is a tree-like structure of recognizers. The structure represents an abstract description of the piece. The description is parsimonious, since it takes into account similarities or identities between one part of the piece and another. It represents the residue, in the mind, which results from listening to a piece of music.

1.2 *Why is this an interesting and important problem?*

The problem of what happens in our minds as we listen to music is clearly an important one, not only because listening is a universal human activity, but also because basic processes involved in listening to music - such as recognizing familiar things or making ways of recognizing new things - is important in any kind of thought.

The world in general may be regarded as presenting us with the problem of constructing useful recognizers for various situations. But music presents us with this problem in a particularly interesting way, because music is deliberately tailored to the mind's cognitive limitations and capacities. Music is constantly involved in a process of selection, in which only that music which successfully addresses these problems survives. The music which has survived this selection process can be regarded as an encoded picture of the mind. This project represents an attempt to better understand this picture by constructing a model, implemented as a computer program, of the processes the mind goes through while listening to music.

1.3 *What is the relation of this project to knowledge-based systems?*

The program is not a knowledge-based system in the sense that one might expect from the description above. The decisions which the program makes, for example, about when to construct a new recognizer are not accomplished by rule-firing or knowledge-based schemes. The knowledge about how to do these things is embedded in the program in the form of "conds" and "ifs."

The knowledge of concern here is the knowledge contained in the recognizers. It is the recognizers which correspond most closely to the rules of an expert system, but they are not rules in the usual sense. They are, rather, records of some bit of music which has been heard, and, implicitly, expectations that that bit of music will be heard again. The program embodies a theory that the musical knowledge acquired by listening consists in such recognizers.

2. How does the program work?

I have described the program as one which listens to music; it would perhaps be more accurate to describe it as one which listens to rhythms, since in the current implementation, the program ignores pitch considerations. It turns out that in simple melodies, pitch and rhythm tend to be redundant, so that one can determine the structure of the piece from rhythmic considerations alone. I believe that extending the program to include pitch considerations is reasonably straightforward, and have left hooks in the code for that purpose.

A note on the organization of this paper: When describing specific code modules, I have included pointers to the appropriate section of code, in the form "(Code References . . .)"

What follows is an event-by-event description of the programs processing of the nursery rhyme "Mary Had A Little Lamb:"

Figure 1.

For each event the program determines whether the event is a good one on which to begin recognizer construction, to end recognizer construction, or both, or neither. The bases for these decisions are the "Grouping Well-Formedness Rules" and "Grouping Preference Rules" outlined in *A Generative Theory of Tonal Music* (Lerdahl and Jackendoff, 1983), tailored somewhat to my purposes. These rules may be summarized as follows:

1. Groups will tend to begin on accented events, and end on the events immediately preceding accented events.
2. We will not form "singleton" groups, i.e., groups consisting of only one event.
3. Events of shorter duration tend to be grouped together with subsequent events of longer duration.
4. A grouping boundary will separate events of longer duration from subsequent events of relatively shorter duration.
5. Groups at the same hierarchical level should be of as equal length as is possible. (The hierarchy is discussed further on in the paper.)
(Code references - (event :startp), (event :endp))

The notes immediately following bar lines are regarded as accented.

Event 1:

We first fire all available recognizers at this event, to determine if it is part of a pattern which

we recognize. Since we haven't constructed any recognizers, however, there are none to fire. We then determine that it is a good event upon which to begin recognizer construction, and start building a new recognizer inside the recognizer-spawner.

(Code references: (recognizer-list :fire-recognizers), (recognizer-spawner :process-event, :start))

Event 2:

Event 2, as it happens, is also deemed a good one on which to start recognizer construction, so we start building a second recognizer inside the spawner. We also add this event to the list of events recognized by the recognizer we started for Event 1, and note that the two recognizers currently under construction conflict, that is, recognize the same bit of the piece. (Code references: (recognizer-spawner :append, :note-conflicts))

Event 3:

The third event is neither a good place to start recognizer construction, nor a good place to end it, so it is simply appended to the two recognizers currently under construction.

Event 4:

This event is determined to be a good one on which to end recognizer construction, so we "spawn" the recognizers currently in the spawner, which is to say that we take them out of the spawner and add them to a list of "newborns." These two recognizers recognize the sections

and

which overlap by more than one note (one-note overlaps are allowed), so we must choose between them. This is generally accomplished by comparing their lengths (in eighth notes) to that of the preceding recognizer, and choosing the one whose length is the least different. Unfortunately, there is no preceding recognizer, but the default recognizer length is the length

of a measure, eight eighth-notes, so the recognizer wins, and the other "dies." (Code

references: (recognizer-list :resolve-conflicts) (recognizer :die)) We now have one recognizer to fire at incoming events.

Event 5:

We fire our single recognizer at event 5, but it immediately fails, since the recognizer's template begins with a dotted-quarter-note, and event 5 is a quarter note. Event 5 is also a good recognizer-starting event, being the first note of the measure and hence accented, so we begin constructing a new recognizer in the spawner.

Event 6

Event 6 is neither a good recognizer starter nor a good recognizer-ender; it is simply appended to the recognizer currently in the spawner.

Event 7:

The seventh event is a good one on which to end recognizer construction, being both relatively long, and immediately followed by a bar-line. Therefore we spawn the recognizer ♩ ♩ ♩ ,

adding it to the list of newborns. The previously constructed recognizer is moved from the newborn-recognizers list to the current-recognizers list. We now have recognizers for

♩ . ♩ ♩ ♩ and ♩ ♩ ♩ .

Event 8:

The ♩ ♩ ♩ recognizer "survives" when it is fired at event 8, which is to say that it recognizes

a match between event 8 and the first note in its template. It increments an internal pointer which will now point to the second note in its template. We can't be sure that it will succeed, however, and event 8 is a good "start-recognizer" event, so we also begin constructing a new recognizer in the spawner. (Code reference: (recognizer :survive))

Event 9:

The ♩ ♩ ♩ recognizer continues to survive on event 9, and we also append it to the recognizer under construction in the spawner.

Event 10:

The ♩ ♩ ♩ recognizer now "succeeds," i.e., completely matches its template. It makes an internal record of its success and resets its pointer. We also destroy the recognizer currently under construction in the spawner, since we don't need two recognizers to recognize the same set of events. (Code reference - (recognizer :succeed))

Events 11-13:

These events are processed in the same way that events 8-10 are.

Events 14-17:

Events 14, 15, and 16 are processed in the same way as in the previous measure (except, of course, that it is the ♩ . ♩ ♩ ♩ recognizer which survives). The success which occurs on

event 17, however, is different from the recognizer successes in the previous measures, because this success is a "resurrection." Resurrections occur when recognizers succeed which have failed on previous firings. Resurrections are important because they signify the return of some thematic material after an intervening absence. This generally indicates a high-level boundary in a piece, and the program uses this opportunity to organize the previously constructed recognizers into a hierarchy of higher-level, or hl-recognizers. We therefore

construct a tree, with the recognizers we have already constructed on the lowest level. In general, a node in the tree recognizes the success of a sequence of child recognizers. These hl-recognizers may be used to detect subsequent sections of the piece, or sections of another piece, which are similar to the section already processed. (Code reference - (recognizer-list :make-hl-recognizers))

Figure 2.

(I have written code to construct the hierarchy, but I have not implemented a system for firing hl-recognizers.)

The next example, a German folk song, is similar to "Mary Had A Little Lamb" in overall structure, but presents the program with additional problems, since there are more overlaps and conflicts between the recognizers, and the groups which are formed are not as nicely aligned with the bar lines.

Figure 3.

Events 1-3:

Since event 1 is a recognizer-starting event, and recognizer 3 is a recognizer-ending event, a recognizer is spawned to recognize these events.

Events 3-7:

Event 3 is both a starting event and an ending event, so a recognizer is spawned to recognize events 3-7, event 7 being the next ending-event.

Event 4 is also a starting event, so we construct the conflicting recognizers $\text{♪} \cdot \text{♪♪♪} \mid \text{♪}$ and

$\text{♪♪♪} \mid \text{♪} \cdot$. We have to choose between them, so we compare their lengths to the previously

reverse of this procedure, varying the rhythm over a fixed set of pitches, is much rarer. This observation contributed to my sense that rhythm is the basic framework which holds music together, and that a listening program should listen, first of all, to rhythm. Having brought the project this far, I believe this was the correct path to take. Adding pitch-processing to the program I have written seems to be a reasonably straightforward task; adding rhythm-processing to a pitch-listening program (whatever that might be) would be much more difficult.

Although the German folk song may not seem, at first glance, much different from "Mary Had A Little Lamb," consideration of the folk song actually resulted in my almost completely abandoning an earlier version of the program. My original idea for the program was to construct recognizers whose boundaries would be completely based on the bar-lines. This happens to work for "Mary Had A Little Lamb," since there the phrase boundaries coincide with the measure boundaries. While there are other pieces in which this phenomena occurs, such an implementation would have been much less general than the current one.

3.2 What does the program know, in relation to the size of the problem space, and how might it be extended?

If we consider the problem space to be all of music, then the program obviously only concerns itself with a tiny fraction of it. On the other hand, the program is not far from being able to handle the space of monophonic rhythms in Western music. This is partly because this space is not very large or complex. Rhythm may be "basic" to Western music, but the "interest" tends to reside in other dimensions. This is not necessarily true of the music of other cultures, and I'm not sure how well such a program would manage, say, Senegalese drum music.

There are several directions in which one might imagine extending such a program. First, there are obvious "holes," such as the lack of any way of firing hl-recognizers, or the fact that the program considers only the rhythmic component of the piece. The lack of a convenient user-interface is another such hole - a convenient language for communicating with this program would be a great boon for further development.

Secondly, there are extensions which would address problems of cognitive plausibility. For example, the current implementation constructs a single recognizer-hierarchy for a given section, and discards recognizers that it can't fit into the hierarchy. Human beings on the other and, seem to be able to maintain several different hierarchies (usually called "interpretations") of a piece. One might consider, therefore, how such a capability could be added to the program.

It sometimes happens that increasing the number of parameters that a cognitive model model considers actually makes the problem more tractable. This was the case for David Waltz, who found that by adding the consideration of shadows to his line-drawing interpretation program, the problem of interpretation was much more tightly constrained (Winston, 1984). This may also be the case for a music-listening program as well. It may be that the additional considerations of pitch and harmony make the interpretation of the structure easier.

All of these extensions mentioned above would greatly increase the amount of bookkeeping involved in maintaining the correct relationships between recognizers. Solution to this as well as the other problems mentioned might be facilitated by use of some frame representation language; I believe however that LISP was the most appropriate choice of language for setting down my initial ideas.

3.3 What did I learn?

The single most important idea which I gleaned from this exercise is the importance of what I called resurrections. It is obvious that the return of thematic material after an intervening absence - as in the recapitulation of a sonata-allegro, or the *da Capo* of an aria - is an

especially meaningful event. The explanation embodied in my model - that this is an opportunity to organize previously acquired bits of knowledge about the piece - is a cognitively plausible one. Furthermore, it is difficult to see how one might have gotten such an idea without being faced with the problem of writing a program which must, at some point, organize that knowledge into a usable form. This is precisely the sort of enlightenment I hoped to gain when I started the project.

There are other lessons from this project which are important but not as easy to pithily describe. The steps which the system goes through - using incoming events to construct new recognizers, checking them against existing ones, deciding which ones to keep and which to throw away, etc., all address problems that any music listener - human or machine - must address. Even a rough solution to these problems yields information about how these tasks can be accomplished.

4. References

The basic idea of constructing recognizers for bits of music comes from Marvin Minsky's 1981 paper "Music, mind and meaning. John Amuedo has written programs to construct parsimonious descriptions of monophonic music. Lerdahl and Jackendoff construct a generative grammar for Western tonal music. Several researchers (Povel, Scott, Lerdahl and Jackendoff) have noted that events which are temporally proximate tend to be grouped together.

Amuedo, J., *Monody* (unpublished work, 1985)

Lerdahl, F., and Jackendoff, R., *A Generative Theory of Tonal Music* (MIT Press, Cambridge, 1983)

Minsky, M., Music, mind, and meaning. *Computer Music Journal*, 5(3) (Fall 1981)

Minsky, M., *The Society of Mind* (Simon and Schuster, New York, 1987)

Povel, D., and Essens, P., Perception of temporal patterns. *Music Perception* 2(4) (Summer 1985)

Scott, D., Duration as a cue to the perception of phrase boundary. *The Journal of the Acoustical Society of America* 71(4) (April 1982)

Winston, P., *Artificial Intelligence* (Addison Wesley, Reading, 1984)