Machine Rhythm:

Computer Emulation of Human Rhythm Perception

by

David Felix Rosenthal

Bachelor of Arts, Mathematics
Master of Arts, Mathematics
Brandeis University
Waltham, Massachusetts
1977

Bachelor of Music
Rubin Academy of Music
Jerusalem
1991

SUBMITTED TO THE MEDIA ARTS AND SCIENCES SECTION,
SCHOOL OF ARCHITECTURE AND PLANNING, IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS OF THE DEGREE OF
DOCTOR OF PHILOSOPHY
AT THE MASSACHUSETTS INSTITUTE OF TECHNOLOGY
AUGUST, 1992

Signature of Author_____
Media Arts and Sciences Section, August 21, 1992

Certified by_____
Marvin Minsky
Toshiba Professor of Media Arts and Sciences

Accepted by_____
Stephen A. Benton
Chairperson, Departmental Committee on Graduate Students

Machine Rhythm:

Computer Emulation of Human Rhythm Perception

by

David Felix Rosenthal

Submitted to the Media Arts and Sciences Section, School of Architecture and Planning, on August 21, 1992 in partial fulfillment of the requirements of the degree of Doctor of Philosophy at the Massachusetts Institute of Technology

Abstract

This thesis describes a computer program, called *Machine Rhythm*, which models the processes by which humans perceive rhythm in music. The program takes as input a polyphonic musical performance represented as a MIDI stream. The output of the program is a complete description of the *rhythmic structure* of the piece: the meter of the piece, the length of the upbeat, and the rhythmic role played by each note.

Machine Rhythm embodies a cognitive model of rhythm perception, and as such represents a tool for investigating intriguing questions about how rhythm perception works in humans. How long does it take a listener to arrive at a rhythmic interpretation? How committed is the listener to an interpretation once it has been constructed? How different are our answers to these and other such questions when we consider music in other cultures? Machine Rhythm represents a framework in which such questions can be more precisely formulated, and answers can be hypothesized and tested. The program also has applications in the world of music technology, such as automatic transcription, automatic synchronization of audio and/or video tracks, and intelligent participation by computers in live performances.

The investigation of *any* human perceptual system raises many questions: Does the brain solve the problem all at once, or does it break it down into parts which are handled by separate agencies? How separate are the agencies involved, and what do they need to know about each other? How are their efforts unified into a single interpretation? and so on. It is largely the lack of good answers to these questions that make problems such as machine vision and audition so difficult. Machine Rhythm gives us a consistent, "working" set of answers to these questions for a moderately sophisticated perceptual system — the system that perceives rhythm — and thereby suggests new approaches to other perceptual modeling problems.

Thesis Supervisor: Marvin Minsky
Title: Toshiba Professor of Media Arts and Sciences

Acknowledgments

In the course of working on this project, I was extremely fortunate to have the support of a number of brilliant, dedicated and generous people. I'll start the list of them with Dan Ellis and Alan Ruttenberg, to whose intelligence, encouragement, and hackertude every part of this work owes a vast debt. It would make me very happy to think that this thesis reflected something of the awesomeness of that team.

I was very lucky to have the help of Marc Davis on the several occasions when it was necessary to put a presentation together. I want to thank Mike Hawley for his invaluable help with testing and providing performance examples. Judy Brown and Tom Knight provided support and sage advice as members of my thesis committee. Daisy and Giok Boen Oey, Mike Travers, David Yadegari, Ken Haase, Barry Vercoe, Janet Cahn, Mary Ann Norris, Jeff Bilmes, Robert Rowe, Greg Tucker, Stuart Cody, Betty Dexter, Betty Lou McClanahan, Molly Bancroft, Tod Machover and everyone in the Music and Cognition group participated in valuable discussions and helped out in other ways.

The most powerful influence in my intellectual world for the past seven years has been Marvin Minsky. His research and teaching convinced me that trying to understand how the mind worked was the most important thing I could spend my time doing, and his support of my graduate studies made spending my time that way possible.

I also want to express my gratitude to my earlier teachers and acknowledge their influence: Viola Haas, Paul Sally, and Haim Alexander.

This thesis owes a great deal to those who had the deepest influences on the ways that I think: Uri Wilensky, Les Faby, Sarah, Judy, Johnny and Felix Rosenthal, and my parents, Patricia and John.

Kim Oey-Rosenthal was my partner in every stage of this work. She provided moral and technical support at every level; her intelligence, understanding, and faith meant that no obstacle had to be faced alone. Good job, Boze.

Finally, I would like to thank my grandfather, Julius Rosenthal, who with love, care, and hope planted the seed that became this work.

Machine Rhythm: Table of Contents

Part I: Preliminary Considerations

Introduction

There's a woman sitting on a park bench, listening to her Walkman, and tapping her fingers in time to the music. It's not an unusual scene, and she's not doing anything particularly amazing. I just have a simple question: what, exactly, is she doing? What are the processes in her brain that cause her to tap when she does, and what relation do they have to the music coming through her earphones?

Well, you might say, it's not all that complicated. She's listening to a recording of a band consisting of four or five instruments; one of the instruments is a drum. When the drummer hits the drum, there's a sharp rise in the energy getting transmitted to her eardrums, and she taps her hand. If it's not a drum it's something else; a loud guitar chord for example.

Except, if we were to try and verify this obvious theory by conducting controlled experiments with microphones, sound-level meters, and recording equipment, and so on, we'd discover that it was wrong. The times at which she's tapping don't correspond to the biggest peaks in the sound energy; in fact, they don't correspond to anything obvious in the audio data. But on the other hand, if we listened to the music ourselves, we would agree that she's tapping in the "right" place; in fact, so would the musicians that produced the music in the first place. It's as though their musical thought was transmitted from their brains to hers in a kind of code that everyone can understand, but no one can give the rules for.

The fact that we can't write down the rules for that code translates into a major handicap when we try to involve computers in music. Simply put, it means that computers, unlike the Walkman-listener, have no sense of rhythm — so they can't transcribe music that's played to them, or participate in live performances as, say, one of the members of that string quartet.

That we can't write down those rules is, in fact, part of a larger problem. In general, we not only don't understand how rhythm works, we don't understand how *perception* works. That's why you have to type to your computer, rather than talk to it, except in a few limited cases. That's also why computers can't drive your car, and robots can't vacuum your floor. In this thesis we'll mainly concentrate on the problem of perceiving rhythm — we'll look at ways to make that code less mysterious. Towards the end, we'll also look at ways to apply what we've learned to the solution of other perception problems.

This thesis describes a computer program, called *Machine Rhythm*, which models the processes by which humans perceive rhythm in music. The program takes as input a polyphonic

musical performance represented as a MIDI stream.[1] The output of the program is a complete description of the *rhythmic structure* of the piece: the meter of the piece, the length of the upbeat, and the rhythmic role played by each note.

Machine Rhythm embodies a cognitive model of rhythm perception, and as such represents a tool for investigating intriguing questions about how rhythm perception works in humans. How long does it take a listener to arrive at a rhythmic interpretation? How committed is the listener to an interpretation once it has been constructed? How different are our answers to these and other such questions when we consider music in other cultures? Machine Rhythm represents a framework in which such questions can be more precisely formulated, and answers can be hypothesized and tested. The program also has applications in the world of music technology, such as automatic transcription, automatic synchronization of audio and/or video tracks, and intelligent participation by computers in live performances.

The investigation of *any* human perceptual system raises many questions: Does the brain solve the problem all at once, or does it break it down into parts which are handled by separate agencies? How separate are the agencies involved, and what do they need to know about each other? How are their efforts unified into a single interpretation? and so on. It is largely the lack of good answers to these questions that make problems such as machine vision and audition so difficult. Machine Rhythm gives us a consistent, "working" set of answers to these questions for a moderately sophisticated perceptual system — the system that perceives rhythm — and thereby suggests new approaches to other perceptual modeling problems.

Why Make A Computer Model?

Why make a computer model of rhythm processing? One answer is that it would be useful for certain applications, such as transcription or automatic synchronization. But there is a deeper reason: that the way to express and test theories of mind is to embody them in a machine.

Theories of mind, unlike theories of physics, do not lend themselves to brief descriptions.

---

[1]MIDI stands for "Musical Instrument -Digital Interface. It is a means of encoding performance information for electronic instruments, usually keyboards. When you play the keys of a MIDI instrument, the resulting signal will contain information about which key was played at what time, how hard it was pressed, and how long it was held. Generally, the performance can be reconstructed from this information. The atomic elements of MIDI are "MIDI bytes" which, for our purposes, can be thought of as either *note-ons* —which contain information about when and how a key was pressed — and *note-offs*, which tell when the note was released.

> My explanations rarely go in neat, straight lines from start to end. I wish I could have lined them up so that you could climb straight to the top, by mental stair steps, one by one. Instead they're tied in tangled webs.
>
> Perhaps the fault is actually mine, for failing to find a tidy base of neatly ordered principles. But I'm inclined to lay the blame upon the nature of the mind: much of its power seems to stem from just the messy ways in which its agents cross-connect. (Minsky 1986, p. 17)

To describe a complicated and dynamic object, we need a precise and expressive descriptive tool, one which can reflect the "messy ways in which agents cross-connect." The programming language, Common Lisp, which I used for this project, is probably not the ideal language for this task, but it's the best we have at the moment.

A computer program derives its expressive power from the fact that it *works.* The workings of an agent, or its effect on other agents, do not need to be imagined; they can be observed. A computer program lends itself to experimentation; a proposed change in the model's operation can often simply be tried out.

Expressing theories of mind as computer programs tends to give us a set of priorities and emphases that are different from those of traditional experimental psychology. Psychological experiments, for example, are often concerned with finding *thresholds* for various effects, such as the minimum difference in pitch that humans can discriminate under various conditions. In a computer model, the threshold might depend in a simple way on a parameter that is easy to change, and its importance would be measured by how it affects the running of the rest of the program.

Experimental psychologists view models as the generators of observable effects that can be measured in repeatable experiments. This is a valuable way of investigating the workings of the mind and verifying theories. Builders of computer models, on the other hand are concerned with *computational feasibility* — they are oriented around what works and what doesn't. Computational feasibility is clearly not a sufficient condition for correctness of a theory of mind — we can write computer programs that do things that minds could never do. It is, however, a *necessary* condition, once we accept that human information processing fits broadly into the same class as that performed by computers.

Building a computational psychological theory often forces the theorist to fill in details that English descriptions of theories omit. A theory of rhythm perception might claim, for

example, that there is a relationship between rhythm parsing and auditory streaming (the organization of sounds based on their perceived origin). So, for example, listeners are able to parse melodies that are rhythmically ambiguous if there is a rhythmically straightforward accompaniment, because the melody and accompaniment are perceived as different sources, and therefore separately processable. But between this statement and a working computer implementation, there are an immense number of treacherous wrong turns and surprising obstacles. The code that negotiates those turns and overcomes those obstacles may seem unimportant to the non-programming theorist, while the programmer comes to regard it as the heart of the problem.

Rhythm Finding Is Difficult

Why is it difficult to program a computer to find rhythms? All we are really doing, one might suggest, is tracking a periodic signal whose period varies with time. There is a known electrical engineering solution for this problem, called a *phase-locked loop.* We will describe a simple device derived from a phase-locked loop, called a *local interval selector* which processes a time-sequence of events, choosing the events which are part of a periodic signal. The selector maintains two state variables, t and Δt, t representing the last time of the last event chosen, and Δt representing the local period of the signal. The selector predicts that the next element of the periodic signal will occur at time t + Δt; the actual event chosen is the one that occurs closest to the predicted time.

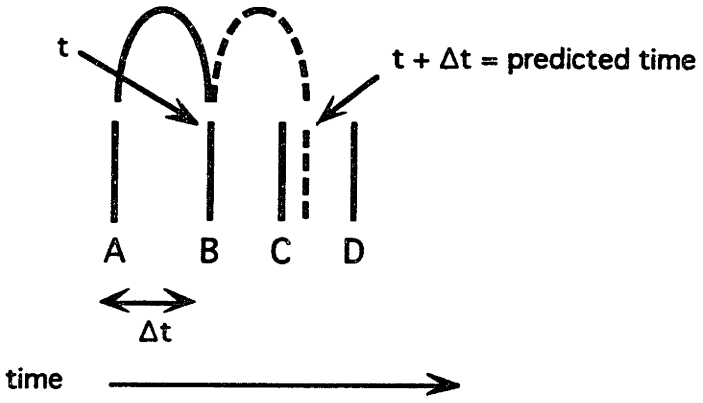


Figure Intro-1. Local interval selector.

In figure Intro-1, events A and B have been determined to be part of the periodic signal, and the selector must decide if the next element of the signal is C or D. Since C is closer to the predicted time than D, C is chosen as the next element. Δt is then updated to be the difference in time between C and B, and t is updated to be the time of C.

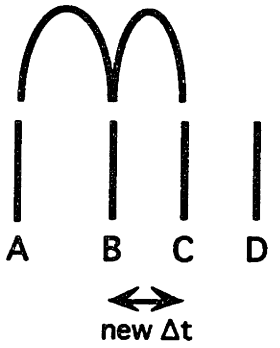


Figure Intro-2.

Why then are local interval selectors inadequate as rhythm finders? There are several problems. One problem is that since their calculations are very local, they are susceptible to "garden path" errors. In figure Intro-3, the loop has been led astray because the sequence (A B C D E) appeared to be the best choice locally, but (A B C′ D′ E′) would have been globally better solutions, in the sense that the overall error (difference between predicted and actual times) would have been less.

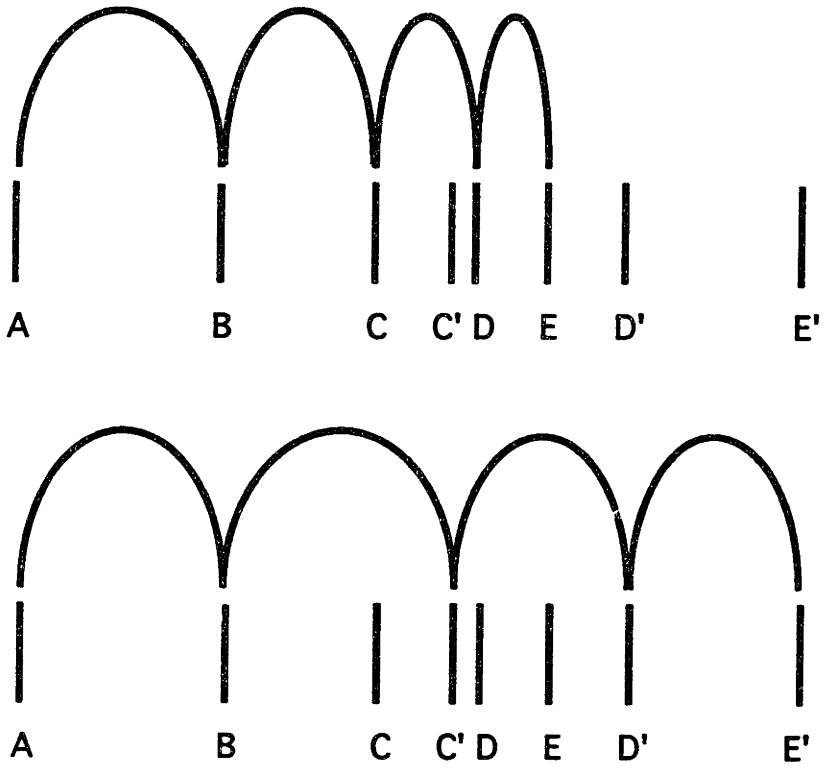


Figure Intro-3.

Another problem is that there may be important considerations that the local interval selector ignores. The local interval selector is choosing the next event solely on the basis of its being "most on time." But in a musical context, other factors besides timing can affect how the next element of a sequence should be chosen. In figure Intro-4, if C is a much more important event than C′ (e.g. a loud chord), C may be a better choice than C′, even though C′ is preferable from the point of view of timing.

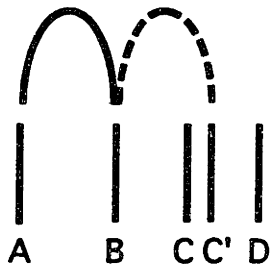


Figure Intro-4.

The possible reasons that C is more important than C′ may be obvious, as when C is much louder than C′, or they may be more subtle, as when C continues a pattern established earlier, or represents a change in harmony.

Perhaps the most important reason why a local interval selector is not an adequate model of rhythm perception is that it only looks at one level of rhythmic activity. Rhythm has a hierarchical structure, where a beat may be subdivided by other beats, and may itself be a subdivision of still another beat. We might better think of a human-like rhythm finder as a set of synchronized local interval selectors whose periods are related by integer factors. We are not arguing that local interval selectors are a bad idea for rhythm-finding. On the contrary, they are a basic building block of the Machine Rhythm system; the system can be thought of as a society of cooperating local interval selectors. The difficulty and the interest of this project lie in the task of managing this cooperation.

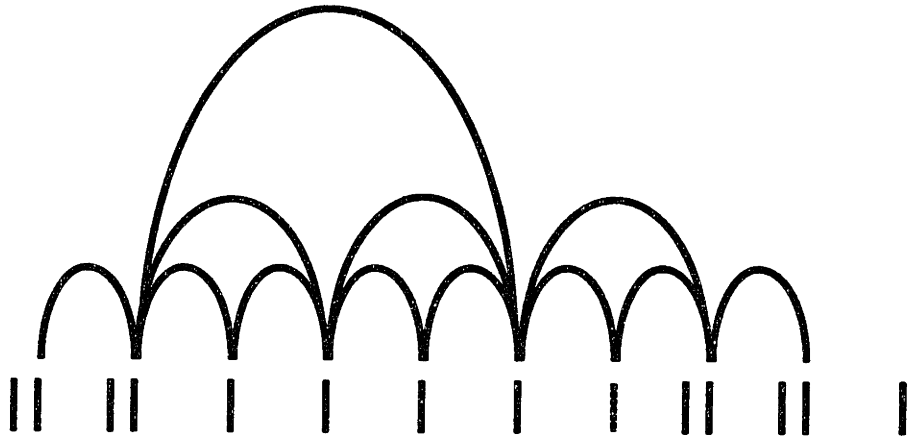


Figure Intro-5. Hierarchy of phase-locked loops.

Some of the points of these last few paragraphs are illustrated in the following example:

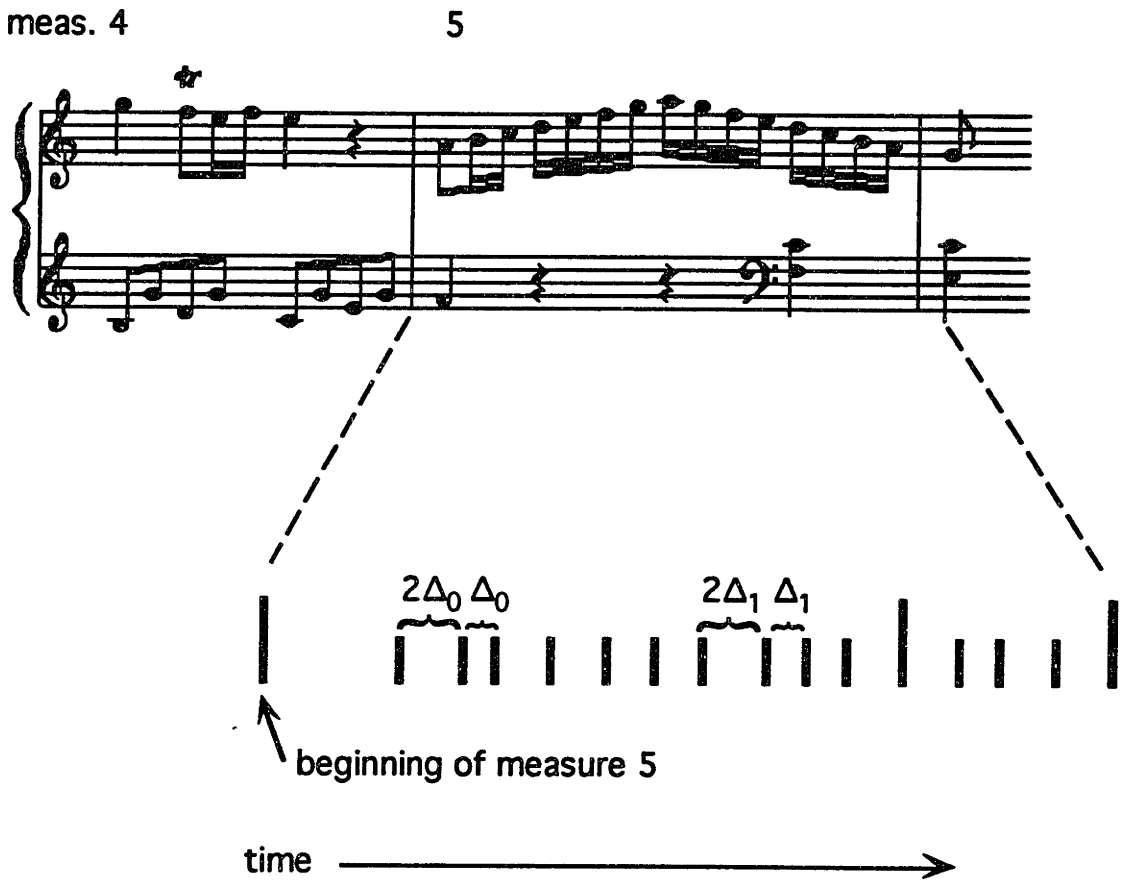


Figure Intro-6. Sloppily played sixteenth-note passage, from Mozart K. 545. The vertical bars indicate note-onsets; longer vertical bars indicate that several notes were played at once, as in the case of the first eighth note in measure 5.

Figure Intro-6 is a graphical depiction of a recording of part of a Mozart piano sonata. The vertical lines show the attack times of a performance of the run of sixteenth notes in measure 5. As the figure shows, some of the sixteenth notes are about twice as long as adjacent ones. Human listeners can hear that the performance is sloppy, but they will nevertheless parse the rhythm of this passage without difficulty. Several characteristics of the performance enable them to do this; the chords which accompany the run, indicated in the diagram by longer vertical lines, cue the listener to the location of the bar-lines, and the change in contour cues the halfway-point between the two bar-lines. These locations serve as convenient anchors for the listeners' perception of the beat; if listeners wander off track, they are resynchronized at the anchor points. In other words, the parsing strategy is more complex, and more flexible, than a simple sixteenth-note counter, which is what a local interval selector would amount to. A machine rhythm-finder must be endowed with a similar flexibility.

If perceiving rhythm is such a difficult task, how is it that we accomplish it so effortlessly? The answer is that the apparent effort involved in accomplishing a mental task is often a poor indication of the actual amount of computation that the task requires. Most people consider solving calculus problems, playing chess, or proving logic theorems to be difficult mental tasks, yet computers have been proficient at all of these things for decades. On the other hand, most people don't consider visually distinguishing between cats and dogs to be particularly demanding, yet it is known that emulating human proficiency at this task on a computer would require vast computational power, and that we do not expect to be able to build such a system any time soon. This should not seem so surprising if we consider that we have, in some sense, been learning to see and hear throughout our evolutionary development, whereas we learn calculus and theorem proving for only a few years. Rhythm perception, like vision, is an ancient system for helping us to make sense of the world. The system has had time to acquire enormous sophistication; we should not expect building a similar system on a computer to be easy.

Problem Specification

The overall characteristics of the Machine Rhythm program: what kinds of input it accepts, what kinds of information it uses, and the kind of output it produces, represent choices from a variety of alternatives. In this section we review the various alternatives and the choices that I made.

Input

Monophonic or Polyphonic?

Most theoretical and computer models of rhythm parsing (e.g. Longuet-Higgens 1984, Lee 1985, Schloss 1985, Povel 1985, and Chung 1989) have assumed monophonic input. Machine Rhythm accepts polyphonic input.

As we will see in subsequent chapters, polyphonic input, requires much more processing and bookkeeping than monophonic input. In exchange for this extra trouble, we obtain two basic advantages. First, a polyphonic system enables us to handle a much wider class of music. Second, a polyphonic source contains much more information than a monophonic source, and much of this information is pertinent to rhythm parsing. It may happen, for example, that the main voice is rhythmically complex and ambiguous, but that the accompanying voice is easy to parse. It is our experience that rhythmic information is provided by the *density* — how many notes sound at once. This kind of information is only available if we are processing polyphonic input.

Stylistic Restrictions on Input

Musical styles and cultures have norms which restrict the ways in which beats may be subdivided and grouped. A common restriction is that beats may only be subdivided into two or three sub-beats. Most Western music conforms to this restriction, which we will call the *binary-ternary* restriction. Another common restriction is that of *isomerism;* we say that a rhythm is *isomeric* if adjacent beats are always divided into the same number of sub-beats. The most common violation of this norm in Western music is the triplet.

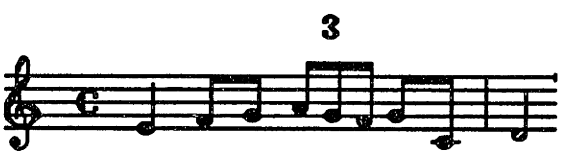


Figure Prob-1.

A related, but different phenomena is that of *meter change*. This refers to a change in the grouping of consecutive beats of the same duration, as shown in figure Prob-2. Meter change is quite common in folk tunes of Bulgaria and Turkey.

Figure Prob-2. Turkish folk song fragment, from (Bartok 1976, p. 149).

In designing a rhythm-parsing program, one must decide what sorts of restrictions on rhythm the program will assume. In making these decisions for the Machine Rhythm program, we follow two principles:

1. Where possible, a restriction should be reflected in a parameter choice. So, for example, the program contains a parameter called *permissible-subdivisions*, which represents the ways in which beats may be subdivided. This parameter currently reflects the assumption of binary-ternary input, i.e., it is set to (2 3). If we wished to explore the consequences of loosening the binary-ternary restriction, we could include, say, 5 in the list of permissible subdivisions.

2. We cannot, at this stage, hope to handle input performances whose style is not in some way restricted, but we want to handle as large a repertoire as possible. Thus we want whatever restrictions we impose to exclude only relatively rare rhythmic phenomena. Non-isomerisms, for example, (especially triplets) are relatively common in Western music, and the Machine Rhythm program is designed to expect them and parse them correctly; meter changes are rarer, and Machine Rhythm does not try to handle them.

Performance Characteristics

Real-time Issues

Machine Rhythm is an implementation of a real-time algorithm; that is, it maintains a notion of "present time," and cannot access information whose time is more recent. The program does not run in real time, in the sense of being able to parse a performance as it is being played; however, this is a problem of processing speed and programming convenience, rather than one of principle. A future goal of this work is to construct a system that can parse a performance as it is being played, and Machine Rhythm is designed with this consideration in mind.

Multiple Sources of Information

Rhythm finding systems differ in the amount of information that they attempt to exploit. Most rhythm parsing models rely exclusively, or almost exclusively, on the *timing* of the events which make up the performance. As we saw in the discussion of phase-locked loops, other attributes of the performance, such as patterns in the melody or accompaniment, changes in harmony, loudness, and so on all contribute to our sense of the performance's rhythm. The problem of parsing is made more complex by including this additional information, and having to integrate the different sources of information into a single result.

The Machine Rhythm program is designed to address this problem. Most of its processing is devoted to making various kinds of clues about the rhythmic structure of the performance available, and unifying them into a single answer. This thesis is in large part an investigation of strategies for accomplishing this integration.

Output

Rhythm finding systems also differ in the output that they produce. Some systems (such as (Schloss 1985) and (Desain 1989)) are essentially *quantizers;* these systems try to remove the tempo variation from the performance. The result can then be input to a system which attempts to construct rhythmic interpretations from exactly timed data. Other systems produce a result that is essentially the number of eighth-notes (or some other appropriate subdivision) that have occurred at a given point in the performance, but without assigning a role to the eighth-notes in any higher level rhythmic structure, or determining what those structures should be.

The view that underlies the Machine Rhythm program is that these various tasks must be integrated. A quantization of the performance into multiples of an appropriate time-interval, or a map of the performance's tempo variation are not the goals of any particular module in the Machine Rhythm program; they are rather side-effects of discovering the piece's rhythmic structure. The output of Machine Rhythm is an account of the rhythmic structure of the performance, which is isomorphic to the description implicit in a musical score. The structure is displayed with a graphical representation like that in figure Prob-3.

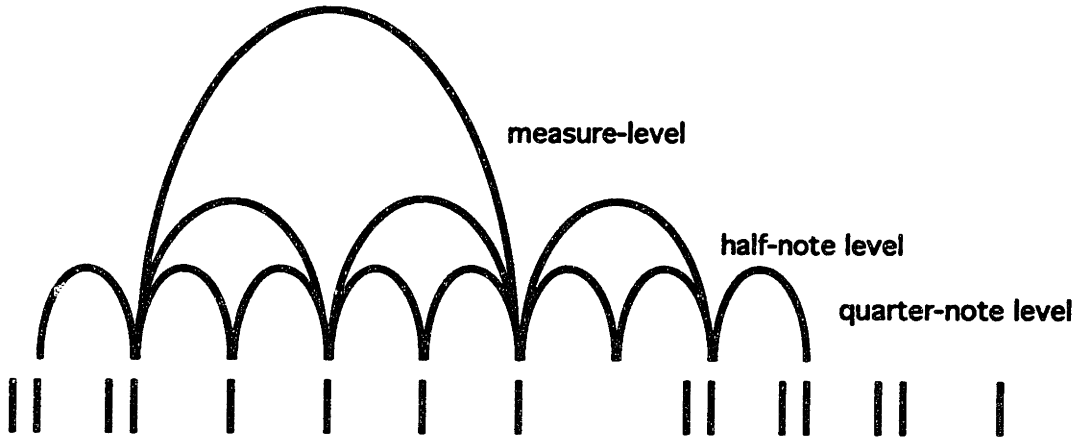


Figure Prob-3.

The vertical bars represent the onsets of notes or chords, where chords, in this context, means sets of notes with roughly simultaneous onsets. The arches represent *beats* — regular pulses in the performance with a certain period and phase. As we see in the diagram, there are several levels of beats. In this example, the smallest arches correspond to quarter-notes, the medium-sized arches correspond to half-notes and the largest arches to measures.

Part II: Program Description

Program Overview

As the program processes the data corresponding to a musical performance, it continually re-represents the data in new forms, and builds new structures. The general trend is a movement of information from stage to stage, each stage reformulating the information and making use of results derived in the stages that preceded it. Thus we start out with a representation whose elements are MIDI bytes[1] representing note-onsets and releases; we progress through representations made up of notes, chords and motives, and finally arrive at representations whose elements are beats and measures. An analogous (though longer and more complex) sequence of events occurs as music enters the ear, is processed into individual events in the auditory cortex, and finally reaches the parts of the mind that "understand music." The program discussed here models neither the beginning nor the end of this process, but an intermediate stretch along the way. The mental processes that this program models are subsequent to the processes that determine that a certain set of pitches is present, but preliminary to the processes that, for example, differentiate Bach from Mozart.

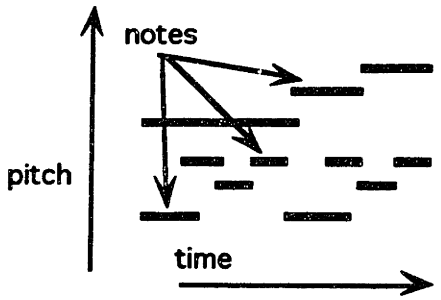
This chapter provides a general overview of the program' s operation, and the structures it creates and uses. Subsequent chapters re-examine many of the ideas introduced here in more detail.

Preprocessing

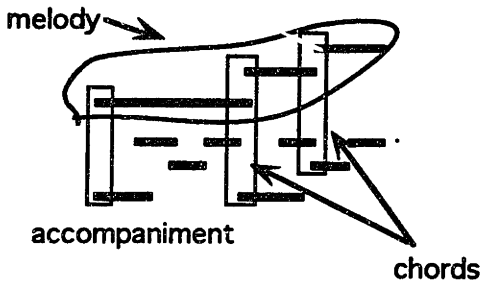
The performance is first analyzed and re-packaged by a preprocessing module.

It is not evident from the representation of the performance as a list of MIDI bytes which groups of notes form *chords* — i.e., perceptually simultaneous groups of notes. Finding chords is the first job of the preprocessor. During the preprocessing stage, we also segregate the events of the performance into *voices*. The rhythm finding routines are then given, as raw material, a list of *summary events*, that is, either chords or single events. Each summary event is composed of one or more *constituent* events, and each constituent event has a tag indicating which voice it belongs to.

[1] MIDI is briefly explained in the chapter "Introduction."



1. The data starts out in the proprocessor as an unstructured list of notes, represented here in a "piano roll" graph.



2. The preprocessor groups the notes into chords and voices.

3. The chords and single events, collectively called *summary events* are represented as vertical bars — longer bars for chords, and shorter bars for single events.

Figure Prog-1. Preprocessing steps.

Startup

After the preprocessing stage, the program selects a set of events from the beginning of the performance, called *startup events*, which are input to the *startup* module. We then create a set of *rhythmic hypotheses* about the startup events. A rhythmic hypothesis is a structure that describes the rhythmic organization of the piece — the time-signature, the position of the bar-lines, and the manner in which measures are subdivided, i.e., the time signature. The graphical representation of a rhythmic hypothesis is shown in figure Prog-2.

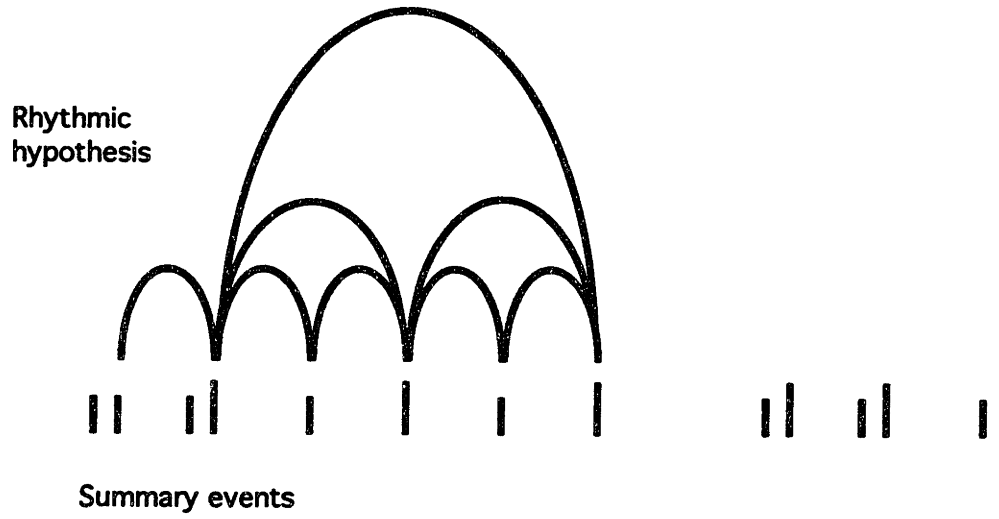


Figure Prog-2. Graphical representation of a rhythmic hypothesis.

The particular hypothesis shown in figure Prog-2 would correspond to the transcription shown in figure Prog-3.

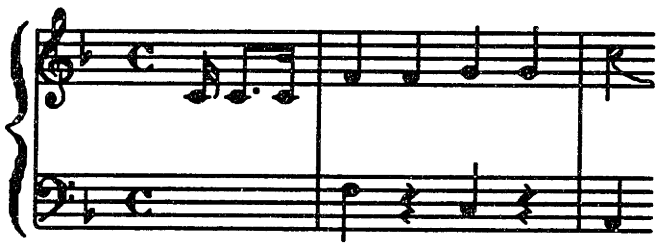


Figure Prog-3. First measure of "La Marseillaise."

Each hypothesis is composed of *beat-levels,* which are in turn composed of *beat-nodes.* A beat node corresponds to the usual idea of a musical beat. A beat node is depicted by a single arch; the *time* of a beat node is indicated by the position of the right side of the arch. Each beat-*level* represents a way of tapping to the music. For the fragment shown in figure Prog-3, for example, one would probably tap the *quarter-note* level; we can think of the quarter-note beat-level in figure Prog-4 as a trace of the tapping finger.

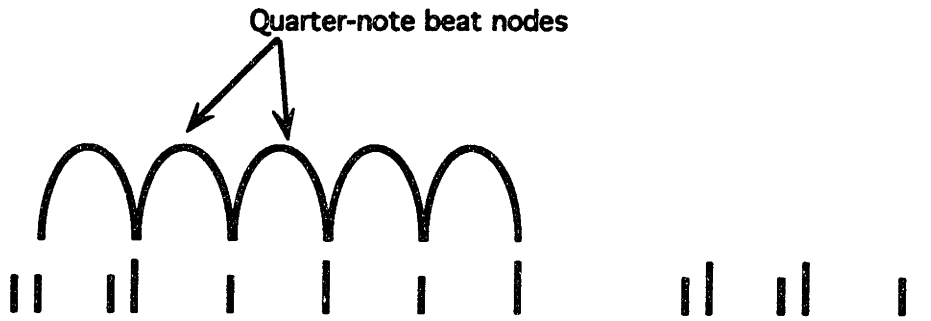


Figure Prog-4

One might also tap the half-notes,

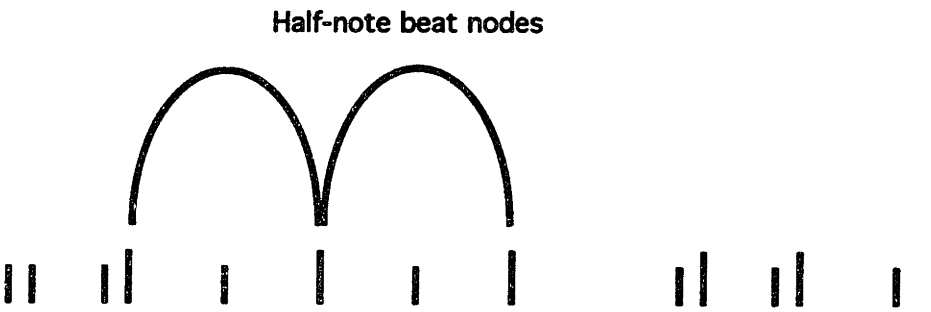


Figure Prog-5.

or, in principle, even the whole notes, or measures.

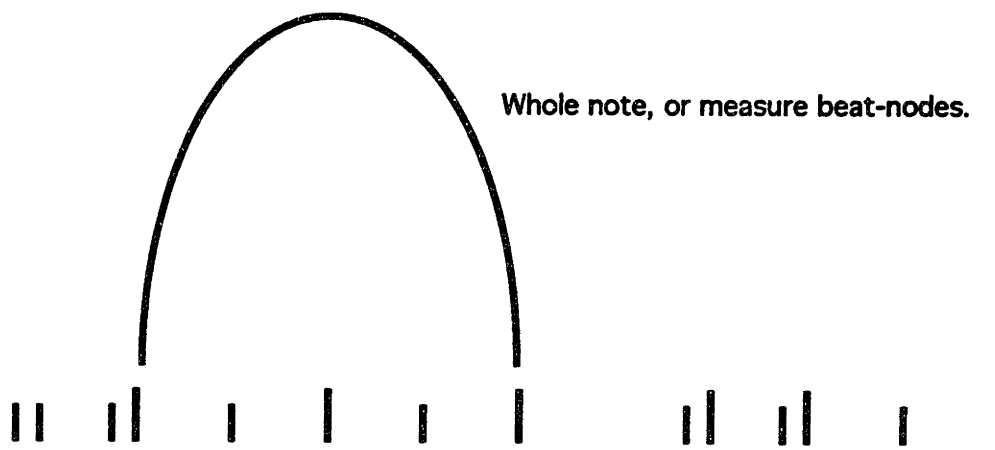


Figure Prog-6.

Different rhythmic hypotheses may vary according to how the beats are subdivided:

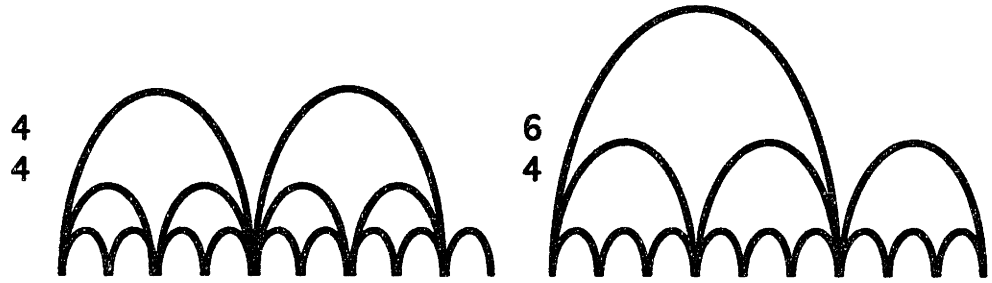


Figure Prog-7. Hypotheses representing different time signatures.

In musical terms, we would say that the two hypotheses shown in figure Prog-7 have different time-signatures, or meters. We can also have two different hypotheses with the same time signature but whose levels are offset by different amounts relative to each other, as in figure Prog-8.

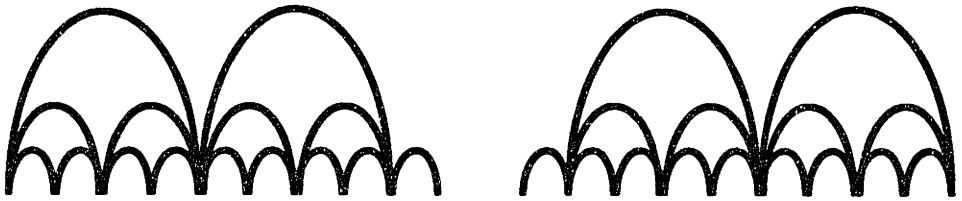


Figure Prog-8. Hypotheses that differ in phase.

We will say that these two hypotheses differ in *phase*. Two hypotheses may have the same time signature and the same phase, but point to different events, as in fig. Prog-9. We will say that these hypotheses differ in *disposition*.

Figure Prog-9 Hypotheses that differ in disposition.

The various hypotheses produced by the startup module are called *progenitors*, for reasons that will shortly become clear. After the progenitors have been created, the *rank* module is called, and the progenitors are ordered according to likelihood that they represent the rhythmic interpretation of the startup events that humans would give.

Extension

As the startup module corresponds to that part of human audition responsible for making an initial guess as to the rhythm of a performance, so the *extender* corresponds to the processes which try to interpret the rest of the performance according to this guess. The extender takes the progenitors produced by the startup module, and extends them to account for the later events of the performance, as shown in figure Prog-10.

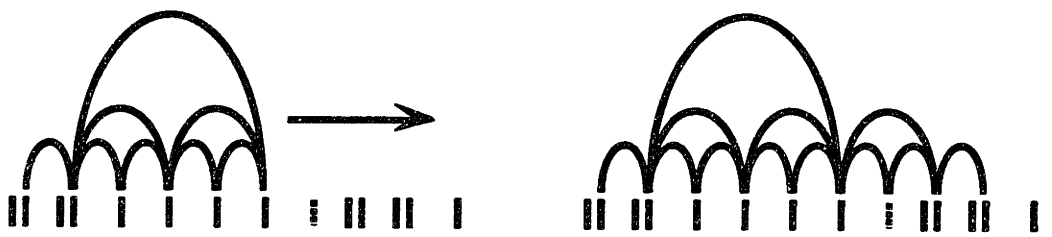


Figure Prog-10. Extending a hypothesis.

As the extender proceeds, it may not be able to decide how the incoming events should be interpreted:

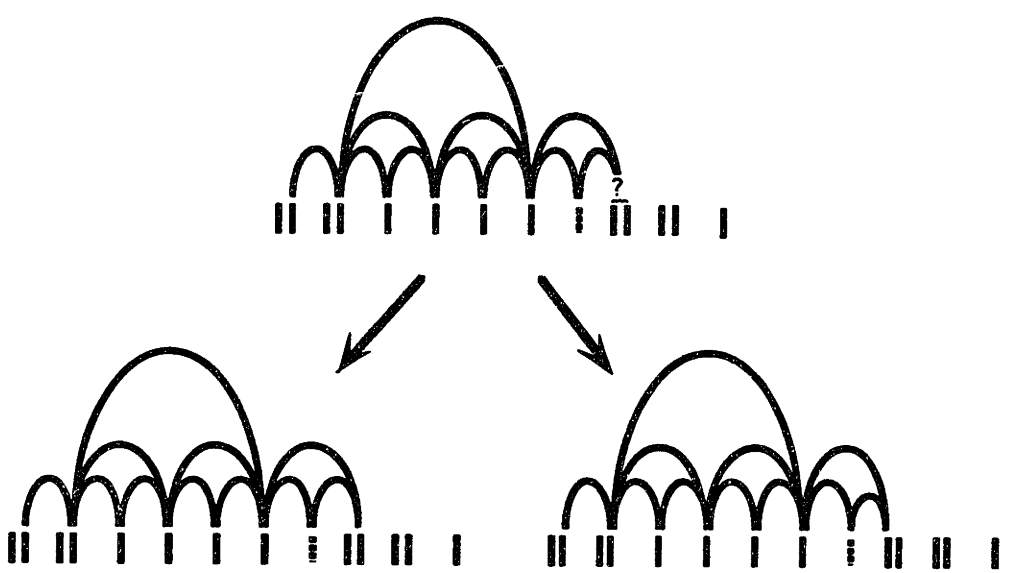


Figure Prog-11. Splitting: the extender can't decide between two different dispositions of the new hypothesis.

When this happens, the progenitor *splits*, becoming several new hypotheses, which can themselves split. The set of hypotheses that split from a single progenitor is called a *family*. The family membership of hypotheses is taken into account in the ranking process, as we will see below.

Alternative Subdivisions

Every time a hypothesis is extended, a module called *alternative subdivision* is called. This module decides whether the division that generates the current lowest level is consonant with the performance, or if it should be subdivided differently. In particular, this module enables the program to detect triplets, as shown in figure. Prog-12.

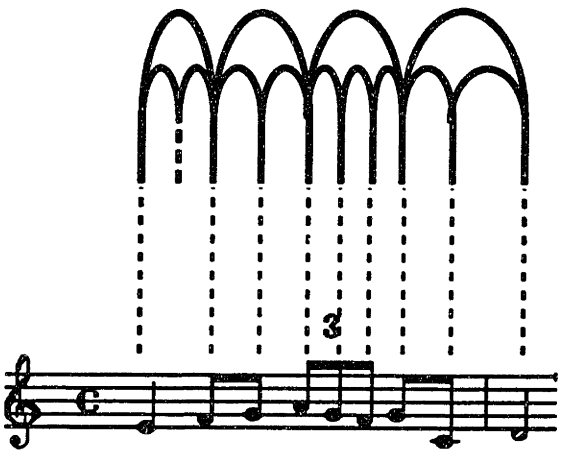


Figure Prog-12.

If the alternative subdivision module decides that an alternative subdivision is warranted, the hypothesis containing the new subdivision is added as a new progenitor— in other words, the resulting hypothesis is not considered a member of the family that gave rise to it. Hypotheses derived in this way are called *mutations*. Note that mutations have different time signatures from the hypotheses from which they are derived.

Ranking

Splitting and mutation cause the total number of hypotheses to increase at a rate which is exponential in the number of events parsed. If this increase is left unchecked, the demands of maintaining each hypothesis quickly overwhelm the computational resources of the system. In practice, we can maintain only a limited number of hypotheses, so we must choose the most promising ones at each stage. The program contains two parameters, called *initiate-pruning-threshold* and *maximum-hypotheses-after-pruning*, which in the current version of Machine Rhythm, are initialized to 30 and 25, respectively. When the number of hypotheses exceeds *initiate-pruning-threshold*, the hypotheses are ranked, and then enough of the lowest ranked hypotheses are eliminated so that the total is less then *maximum-hypotheses-after-pruning*.

Ideally, the correct hypothesis — the one that humans choose — is ranked first. Failing that, the rank module should at least give the correct hypothesis a sufficiently high rank so that it is not eliminated by pruning. It often happens that a rhythmically complex passage will cause the correct hypothesis to temporarily slip from first place, and return to first place when subsequent events make the correct interpretation more obvious.

Preprocessing

This chapter describes the preprocessing module, whose main job is grouping the performance data into chords and voices.

From MIDI Bytes to Note Events

The first preprocessing step is to simply associate the corresponding note-ons and note-offs in the MIDI data. The performance is then represented as a list of objects called *note-events*, which can be represented graphically in the "piano-roll" style shown in figure Prep-1. Each note event contains information about the note's onset time, pitch, loudness, duration, and inter-onset-interval (henceforth IOI), or time to the next note onset.

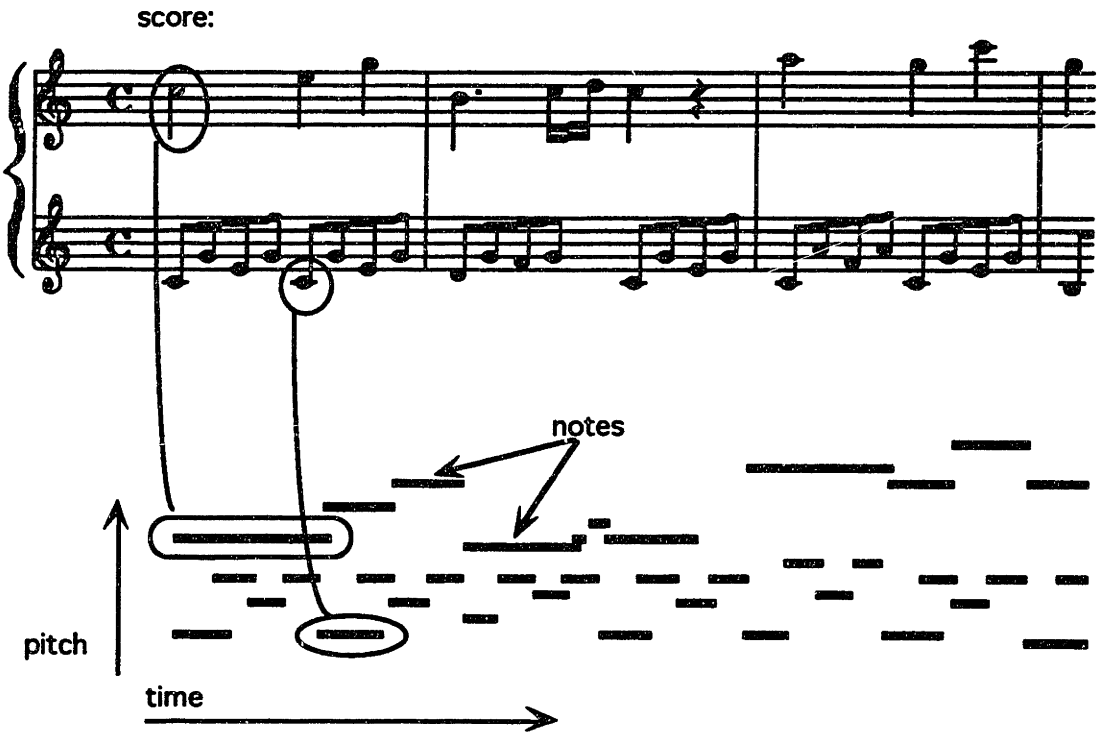


Figure Prep-1. Score of the first three measures of Mozart Sonata K. 545, and the corresponding piano-roll representation.

Finding Chords

The second step is the formation of *chords*. For the purposes of rhythm-finding, we employ a heuristic definition of a chord as a group of events which are judged by the listener (and presumably intended by the performer) to be simultaneous. This is necessary because onset times of notes which are played "simultaneously" often differ by more than 10 milliseconds, the

temporal granularity (usually) of MIDI. Before this step, the representation of the performance is a list of single notes; after this step it is a list of summary events, where each summary event is either a single note or a chord.

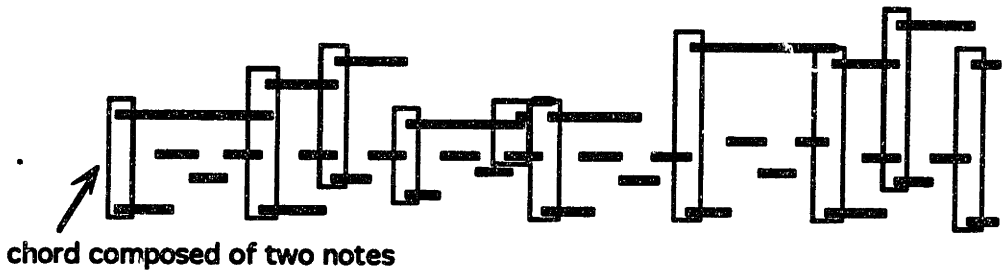


Figure Prep-2. Chords — sets of notes deemed effectively simultaneous — are indicated by vertical rectangles in the piano-roll representation.

The process by which Machine Rhythm produces summary events is a simplification of a more elaborate process that takes place in the human auditory system. Listeners group sounds at several different levels, and construct hierarchies of the groupings. A group of notes may be considered a single unit for some purposes, and a sequence of separate events for other purposes. Thus the notes of what we call a chord may be sequentially distinguishable.[1] It is important to keep in mind that *our* ultimate goal is to find the rhythm; therefore our concern is whether the separation between two events is *rhythmically* important. More precisely, the questions before us are: is the time between the two events the interval of a beat at some level, or does the group containing the two events correspond to a single beat?

Our chord finder is based on experimental work by Van Noorden and others, summarized in (Bregman 1990). The results summarized by Bregman do not bear directly on the question of how to find chords; they concern, rather, the process of *stream segregation*. Bregman describes experiments of the following sort: the listener is presented with a sequence of tones whose pitches vary as shown in figure Prep-3.

[1] (Bregman 1990) and others (Palmer 1988) have pointed out separating a melodic lines from accompaniments is easier when they are slightly asynchronous, by about 30-50 milliseconds.

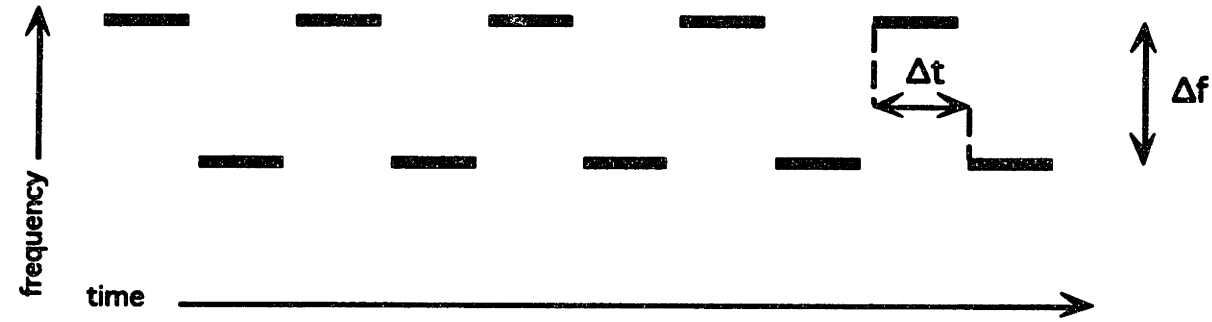


Figure Prep-3

The tones alternate between two pitches separated by Δf, and their onsets are separated by Δt. This stimulus is interpreted by listeners in one of two ways: either they hear a single continuous stream of tones that alternate between the two pitches, or they hear two separate streams of tones:

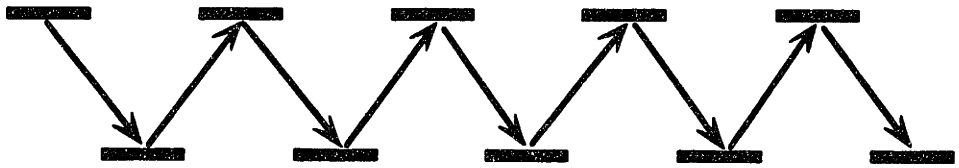


Figure Prep-4. The stimulus in figure Prep-3 may be heard as a single stream, . . .

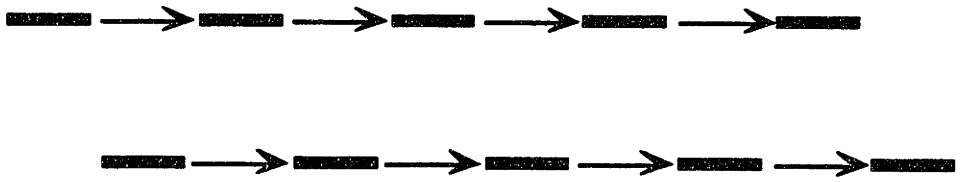


Figure Prep-5. . . . or as two separate streams.

The interpretation that listeners choose depends both on the difference in pitch between the two tones and on the amount of time separating their onsets. The more time between onsets, the greater the likelihood that the listener will hear a single stream; also, the likelihood that they will hear a single stream increases as the tones become closer together in pitch. The results of these experiments are summarized in the following graph:

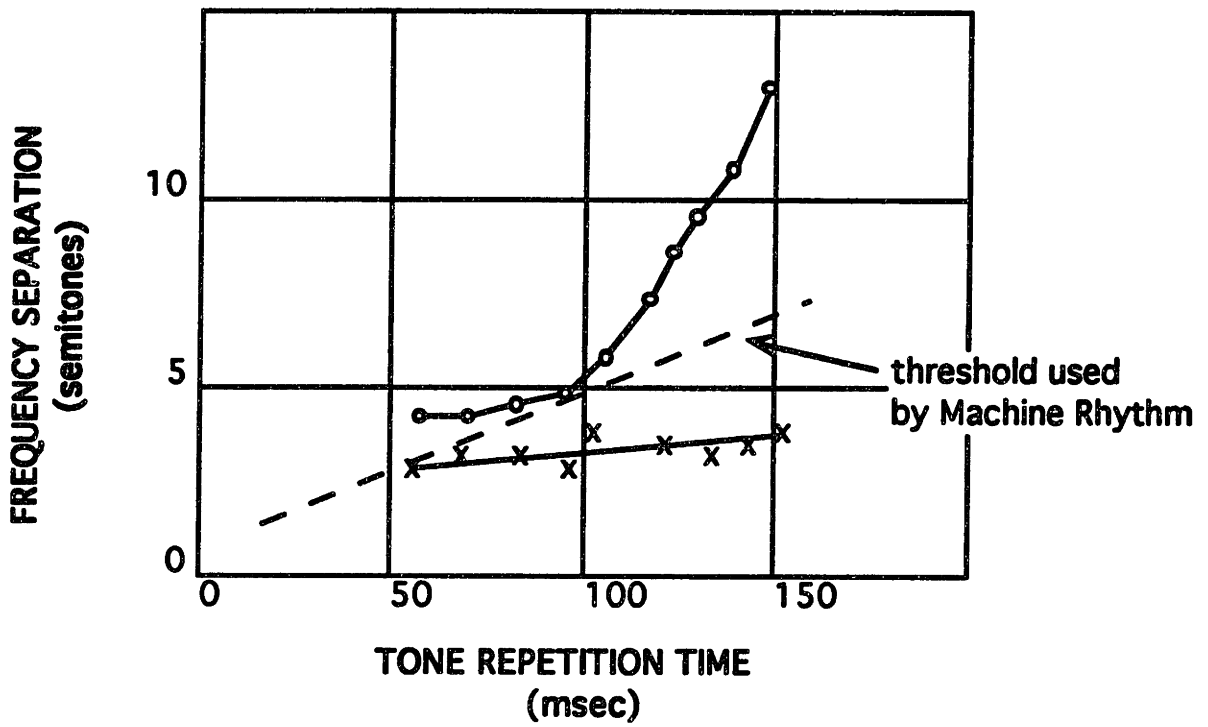


Figure Prep-6. Graph of the "trill threshold," from (Bregman 1990, p. 60). Subjects are presented with a stimulus like that shown in figure Prep-3, which they hear for certain Δf and Δt, as a single stream (or "trill"), as in figure Prep-4. When Δf becomes sufficiently large, they begin to hear two separate streams, as in figure Prep-5. The threshold at which this occurs is the *temporal coherence boundary*, and is indicated by o's. In another experiment, subjects are presented with a stimulus which they hear as two separate streams, as in figure Prep-5; Δf is gradually narrowed, until subjects begin to hear a single stream. The threshold for this effect is the *temporal fusion boundary*, and is indicated by x's.

When the separation in frequency and time of two notes is below the dotted line in figure Prep-6, we will say that the notes are *within the trill threshold;* otherwise they are *outside the trill threshold.*

To incorporate these results into our model, we make the following assumptions:

1. Chord formation will follow the same principles as stream formation with regard to inter-onset-interval and pitch. Less time between onsets and greater difference in pitch will make interpretation as a chord, rather that as two sequential events, more likely.

2. The results shown in figure Prep can be used as an appropriate guide to the parameters of our chord finder — i.e., Machine Rhythm can use the threshold indicated by the dotted line in figure Prep-6.

We also have to make some generalizations, because chords in the performances that we want to be able to deal with can contain an arbitrary number of notes, and the auditory streaming results only tell us what to do in the case of two. In particular, we must deal with the situation shown in figure Prep-7.

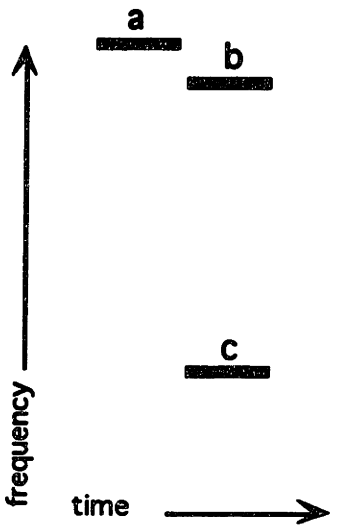


Figure Prep-7.

In this example, either one of the pairs (a, c) or (b, c) are outside the trill threshold, that is, they are close enough in time and far apart enough in frequency to be considered part of a chord. The notes a and b, however, are within the trill threshold, and must be heard sequentially.

We will handle this case by generalizing the experimental results summarized in figure Prep-6 to the following algorithm:

1. Find groups of notes that are separated at the start of the group and at the end of the group by pairs of notes that could not be part of a single chord.

2. Find the "worst pair" of notes in the group, the pair of notes that is furthest inside the trill threshold (in other words, the distance from the point in the graph of figure Prep-6 representing the pair to the line representing the threshold is greatest). If the worst pair is outside the trill threshold, we're done.

3. Construct two candidate chords by excluding each of the notes in the worst pair found in 2. One of these chords will be better than the other, in that its worst pair will not be as bad as the other's worst pair. This becomes the new candidate chord. Go to step 2.

Consider how this works on the chord shown in figure Prep-7. Suppose (a b c) is our original candidate chord. Its worst pair is (a b), so we construct the candidate chords (a c) and (b c). Of these, (a c) is worse than (b c), so (b c) becomes our new candidate chord. Since b and c are within the trill threshold, we're done.

Separating Voices

The third preprocessing step corresponds to the process of *auditory stream segregation.* In this step the program attempts to segregate the performance events into voices.

First of all, why do we want to separate voices? The reason is that a strong clue to the rhythmic structure of a piece is often provided by a repeating melodic pattern. Consider the following example:

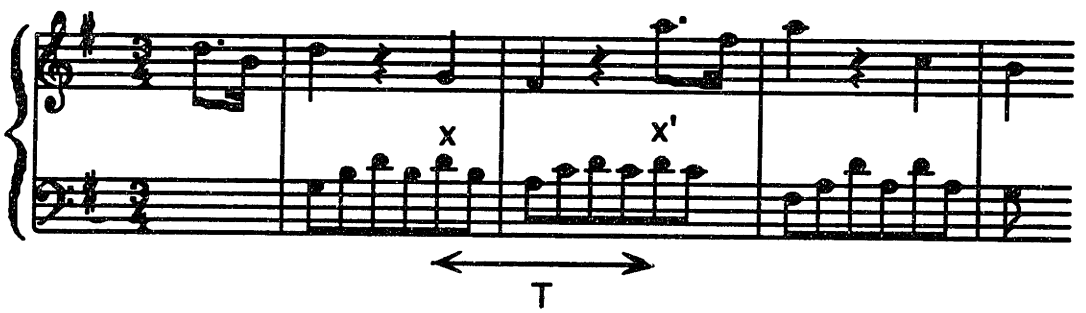


Figure Prep-8.

Here, humans are strongly clued to the fact that measures are made up of six eighth notes by the repeating six-note bass pattern. This allows us to eliminate rhythmic interpretations which are not consonant with this pattern, such as that shown in figure Prep-9.

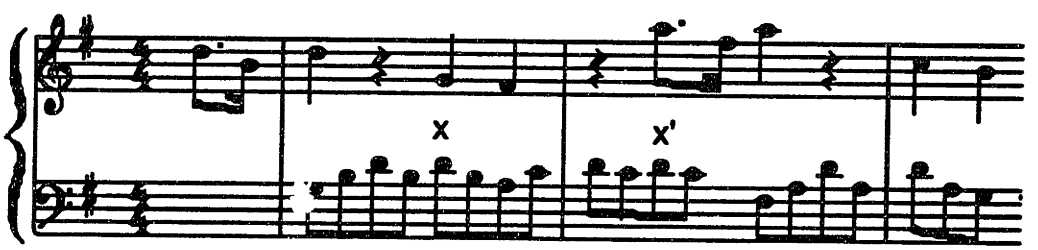


Figure Prep-9.

For the program to use this information, it first has to locate the pattern. Although the pattern is easy to detect with our eyes or ears, we have little insight into how or why we detect such patterns, and so to write an efficient computer program to do it is difficult, and the resulting

program will be expensive in terms of computational resources. It is obvious to us, for example, that the notes x and x' in figure Prep-8 correspond. But how will the program know that? It must guess that the offset between x and its corresponding note is T. Even then it's not done, since the corresponding note could be any of the notes occurring at the appropriate time slot. By knowing that some of the notes at the time slot in question are not under consideration, because they are in a different voice from x, we can make the pattern-finder's job easier. Even if the melody separator mistakenly classifies some of the notes, it can still provide useful information to the pattern finder.

The procedure for separating the melody is as follows:

1. Consider the first summary event (note or chord) of the performance. If it's a chord, classify the top note as melody and the remaining notes as accompaniment. If it's a single note, classify it as melody.

2. Set the *melody-candidate* to the next summary event if that event is a single note, and to the highest-pitch chord-constituent if the event is a chord.

4. Check to see if the melody-candidate is the highest-sounding event for most of the time that it sounds (the "top-reasonably-clear" criterion). If this is not the case, add the event to the accompaniment, and return to step 2.

3. Otherwise, add the event to the melody if either of the following criteria hold:

   (a) The note is closer to some recent melody event than to any recent accompaniment event.

   (b) A long time has passed since the end of the last melody note.

Repeat step 2.

"A long time" in step 3a means longer than a threshold established by an appropriate parameter; "recent" in 3b is similarly defined. "Closer" in 3b refers to distance in a two dimensional Euclidean space whose vertical dimension is pitch and whose horizontal dimension is time.

We note that there are two different metrics that we are using to group events. When we are grouping events into chords, we regard events as "closer" (that is, more likely to be part of the

same chord) when, for a given separation in time, they have pitches which are more different. When we are grouping events into voices, we regard events as closer (that is, more likely to belong to the same voice) when the pitches are less different. This view is consistent with the results outlined in (Bregman 1990).

We also note that Machine Rhythm's voice separator makes the following assumptions:

1. The piece can be thought of as consisting of a monophonic melody line and an accompaniment.

2. The melody is always higher in pitch than the accompaniment.

Although it is not difficult to find examples where these assumptions are unwarranted, we should remember that the goal of this process is not to construct a complete and accurate model of the auditory streaming process, but rather to make the pattern finder's job easier. Most of the time, the melody *is* in the uppermost voice. As with the parts of the program that depend on the chord finder, the pattern finder must be robust enough to compensate for the fact that the voicing process will not always produce meaningful results.

Summary

The input to the preprocessor is a list of MIDI bytes. The processes described in this chapter transform this representation into one where it makes sense to talk about *chords*, *notes*, and *voices*. Although the input to the Machine Rhythm program is polyphonic, we are modeling rhythm as a monophonic process.[2] This means that we must create an appropriate monophonic stream from the polyphonic input data. This is the job of the *chord finder*. The output of the chord finder is a list of *summary events;* the list of summary events will be the input to the main rhythm finding modules.

We also observe that there are important clues to the rhythmic structure of the performance in the *patterns* that may occur in the melody or the accompaniment. Finding these patterns is a combinatorially explosive task. Human listeners are aided in this same task by the fact that they have sophisticated processes which perform *auditory streaming*, which greatly reduces

---

[2] It can be argued that humans can, under some circumstances, create several separate and simultaneous rhythmic streams; the musical term for this is *polyrhythm*. Our model does not, however, attempt to model this process. Note that this is a separate question from whether the *input* to the program is polyphonic. Polyphonic music is quite common; polyrhythmic music — music where, for example, one voice is in (4 4) and another is in (3 4) — is much less widespread.

the amount of search the pattern matcher has to perform. We attempt to give our program a similar advantage by giving it some ability to segregate voices.

Startup

In the startup phase, the program does not yet have any information about the rhythmic structure of the performance. Consequently, we make somewhat stronger assumptions about the rhythmic regularity of the beginning of the performance than are required once the rhythm is established. In particular, we assume that the beginning of the piece is isomeric (i.e., no triplets in a duple rhythm), that the tempo is initially relatively constant, and that syncopations are relatively infrequent.

The Tactus

The first step in the startup procedure is determining an interval called the *tactus*. In musical terminology the term "tactus" means, roughly, the basic beat of the music, or the rate at which a listener would be inclined to tap, usually corresponding to a quarter-note or eighth note. After determining the tactus, we will build the *tactus-level* — the rhythmic level whose interval is the tactus. Other rhythmic levels will be determined by their relation the tactus level.

The process for determining the tactus resembles a procedure developed by Minsky. and appropriated by Gold and Rabiner for tracking pitches in audio signals (Gold 1969). The first step is to make a histogram of the time differences between every pair of events in the set of startup events. We will call the dependent variable of the histogram p, so that p(i) is the popularity of the interval i. We then compute the *harmonic popularity* $p_h(i)$,given by

$$p_h(i) = p(i) + w_2 p(2i) + w_3 p(3i),$$

where $0 < w_2 < 1$ and $0 < w_3 < 1$. The perception of beats at spacing i will be reinforced by events spaced at multiples of i. Consider the following example:

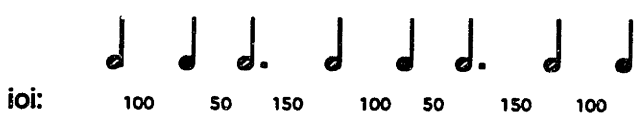


Figure Start-1.

Suppose that quarter-notes in this passage are 50 centiseconds long. Then the interval 50 (quarter-notes) will have popularity 2, interval 100 (half notes) will have popularity 3, and interval 150 (dotted halves) will have popularity 2. But despite the fact that interval 100 is the most popular in this passage, the tactus is clearly 50; we have no choice but to hear the half-notes and dotted-half-notes as divided into quarters:

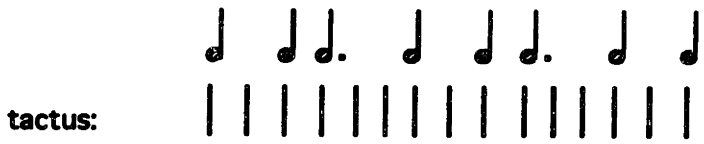


Figure Start-2.

This is because the half-notes and dotted half notes reinforce the perception of a beat whose interval is their least common denominator. In other words, the occurrence of an interval encourages the perception not only of itself as the tactus, but also of beat-levels which subdivide the interval. The parameters w_1 and w_2 control the magnitude of this effect. With w_2 and w_3 each set to 1/2 in the example from figure Start-1, we obtain $p_h(50) = p(50) + p(100)/2 + p(150)/2 = 2 + 1.5 + 1 = 4.5$.

Figures Start-3—Start-5 show how we determine the tactus with data from a recording of a Mozart sonata.

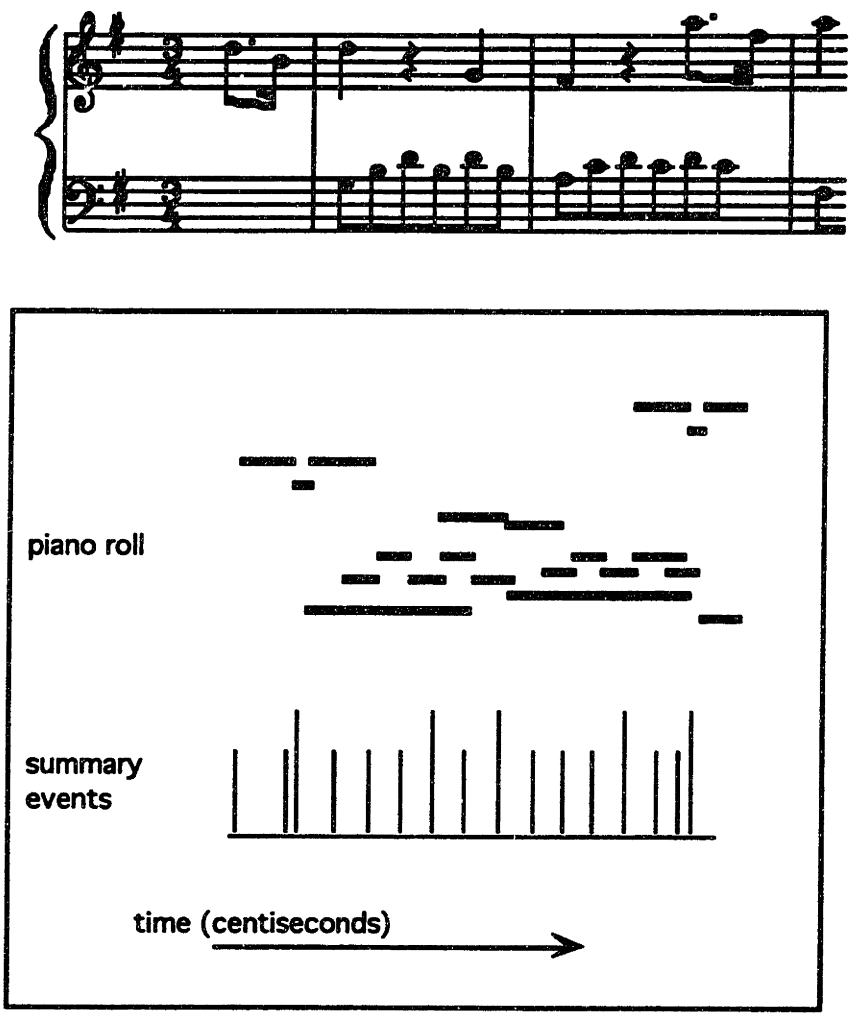


Figure Start-3.

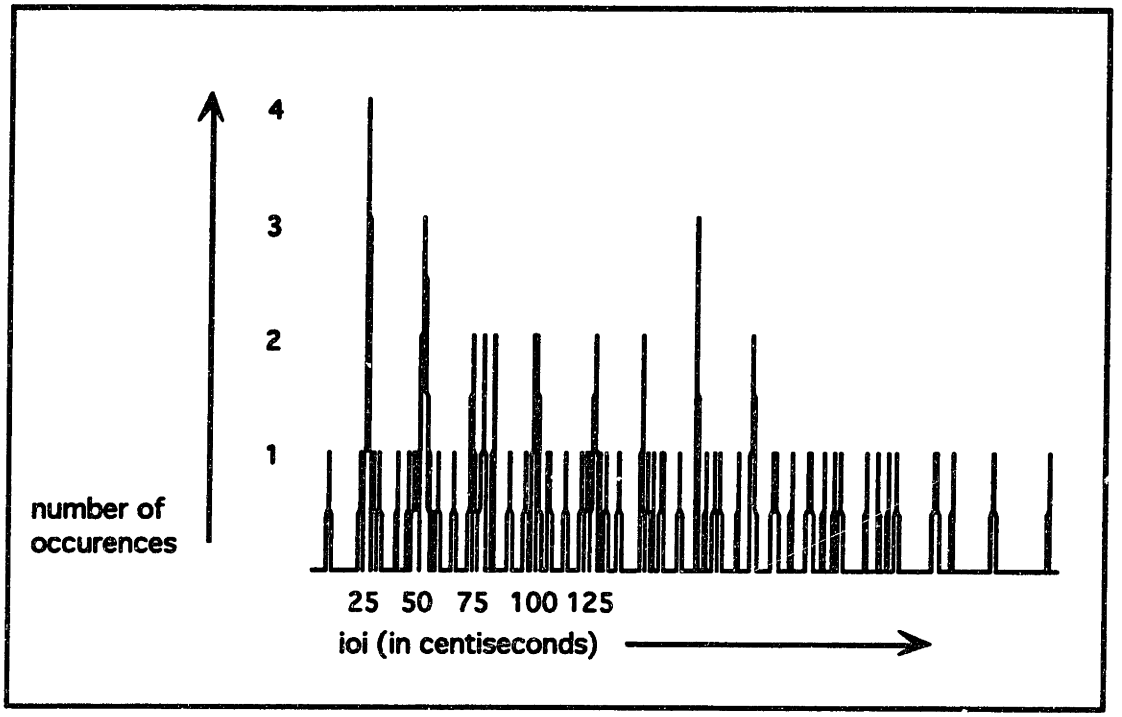


Figure Start-4. Harmonic popularity of IOI's of pairs of startup-events.

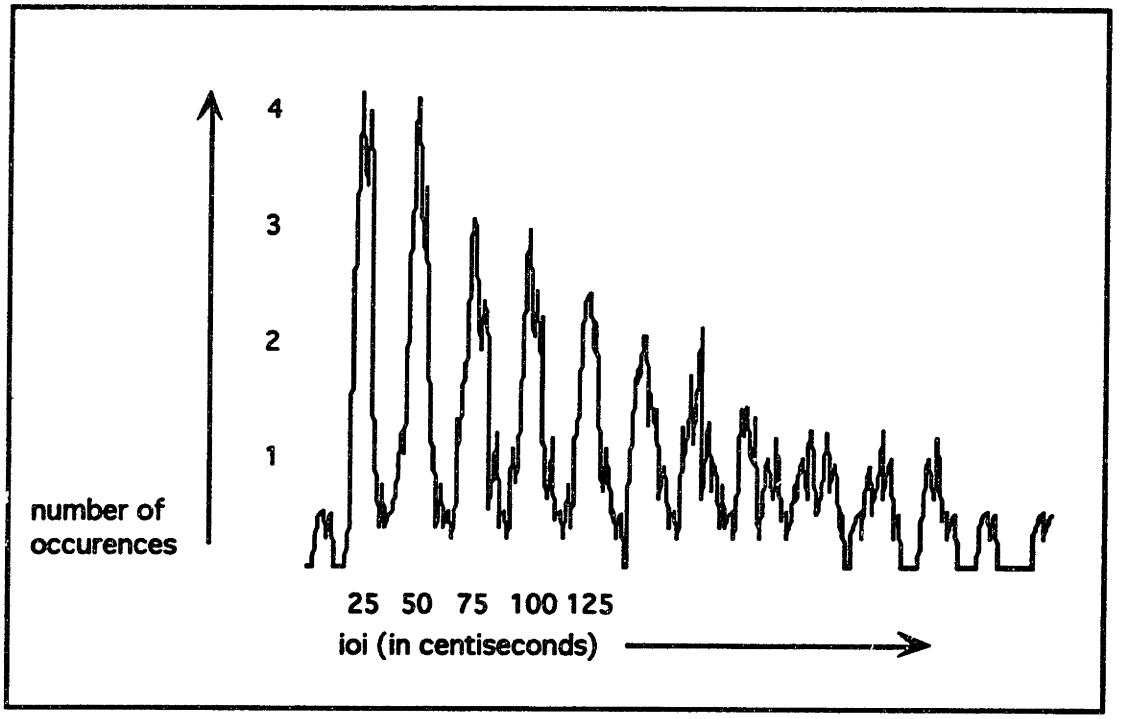


Figure Start-5. Graph from figure Start-4 after smudging with a Gaussian filter.

Figures Start-4 and Start-5 are a graphs of harmonic popularity for the fragment shown in figure Start-3.

After calculating the histogram of the IOI's we smudge it with a Gaussian filter. The effect of smudging is that the smudged popularity of a particular IOI will depend not only on how many times that IOI occurred, but also on how many times IOI's in its neighborhood occurred. This is an appropriate step when we expect the IOI's that represent the tactus to vary in the neighborhood of some value.

The tactus interval is then determined by the formula

$$\max_i \{p_s(i)\left(1-\frac{i}{i_{max}}\right)\}$$

where $p_S(i)$ is the (smudged) number of occurrences of the interval i, and i_{max} is the maximum IOI considered. The effect of this formula is to bias the choice toward smaller IOI's; if, for example, two IOI's are equally popular, we will choose the smaller of them.

The Tactus Level

The next step is to construct a rhythmic level whose interval is the tactus.

To do this, we consider each event in the list of startup events, in order of increasing time. For each event:

1. Construct a beat-node which points to the event and whose interval is the tactus.

2. Calculate the *ideal next time:* the sum of the event's time and the interval.

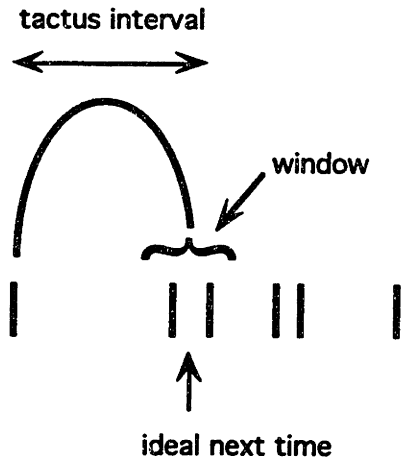


Figure Start-6.

3. Collect all the events whose times are in a window surrounding the ideal next time. If there are none, create a ghost-event at the ideal time. If we have already created ghost-events, and creating another one would cause us to exceed *startup-maximum-successive-ghosts*, start over at step 1 with a new event.

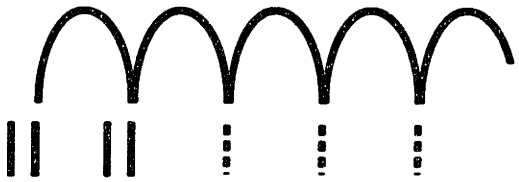


Figure Start-7. Too many ghosts. Ghost-events are indicated by dotted lines.

4. If there were events in the window in step 3, call the function *event-chooser* with the candidate events as arguments. Event-chooser attempts to select the event most likely to constitute the beat that follows the beat established in step 1.

5. Create a new beat-node which points to the event chosen in step 4, or the ghost-event chosen in step 3, and go to step 2.

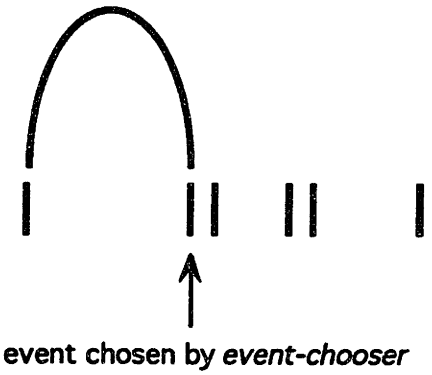


Figure Start-8.

We have not shown how the window in step 3 is calculated, nor have we said how the *event-chooser* module works. Both of these topics are covered in chapter "Choosing and Ranking."

It can also happen that the algorithm described here will produce several different possibilities for the tactus level. This happens for two reasons: first of all, the algorithm may terminate several times at step 3, producing isolated "islands" where the tactus is determined; also, it sometimes happens that *event-chooser* cannot decide between two possibilities, producing the situation illustrated in figure Start-9 In this latter case, we say that the two tactus levels have different *dispositions*. When there is more than one possibility for the tactus-level, we filter out any that are

obviously unsuitable (such as tactus-levels that contain only two or three beat-nodes) and pass the rest along to the next stage.

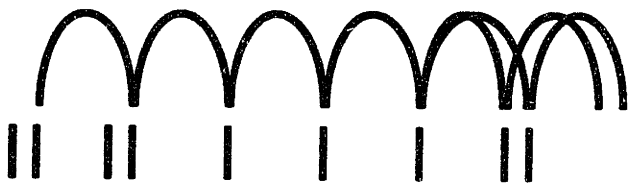


Figure Start-9. Two tactus levels with different dispositions.

Building Hypotheses

The next step is building hypotheses — in other words, determining all of the ancestor-levels of the tactus level. Assuming that only binary and ternary subdivisions are allowed, a tactus will have five possible parents: two binary and three ternary, where each alternative parent with the same subdivision factor represents a different phase. Each of these parents, in turn, will have 5 parents, and so on; we build ancestor levels until the largest exceeds the parameter *measure-size*. Part of the process of constructing the ancestor levels for a tactus level is illustrated in figure Start-10.

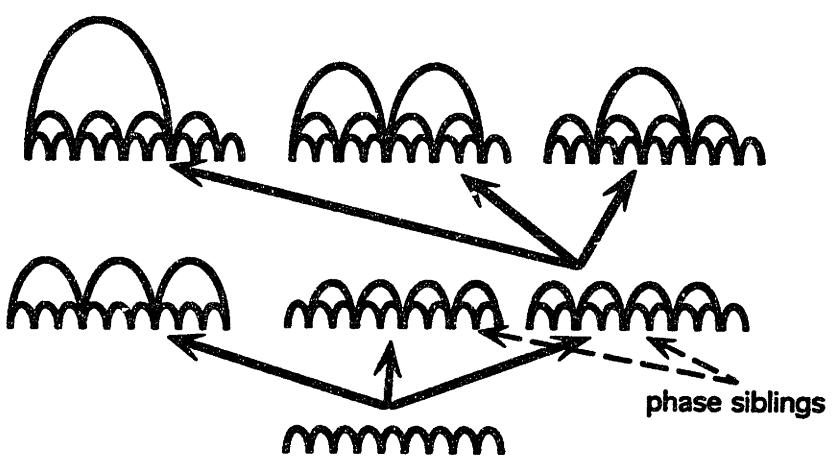


Figure Start-10. Some of the ancestor levels for a tactus level

The total number of possible hypotheses, for a given tactus level is lp^n,

where l is the number of different tactus levels, p is the number of parents (in the case we are considering, 5), and n is the number of levels above the tactus level. For typical values of l, p, and

n, this number can easily exceed 100, which is more hypotheses than the system, as currently configured, can comfortably handle; we need to keep the number of hypotheses to about 10 or less. One way of doing this is to choose one representative from each set of *offset siblings* — parents with the same subdivision but different offsets (see figure Start-10). Our first task, then is to select the best parent from each set of offset-siblings.

The discussion of the procedure for choosing the best offset-parent of a level together with the discussion of the event-chooser, is taken up in the chapter "Choosing and Ranking."

At the end of the startup process, we have a collection of hypotheses for each tactus level, each having a unique time-signature, as shown in figure Start-11. These hypotheses, for reasons that will become clear in the chapter "Extending Hypotheses," are called *progenitors*.

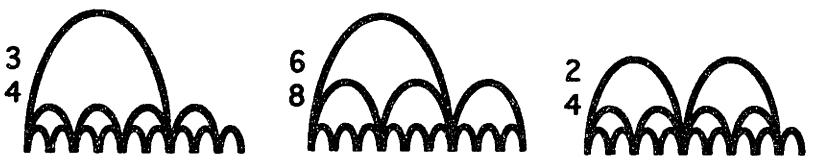


Figure Start-11.

Summary

The startup procedure produces a set of hypotheses about the rhythm of the beginning of the piece. Its first step is to find the tactus level; having found one or more tactus levels, it then constructs a set of hypotheses for each. This set is exhaustive, in the sense that it includes (with the stated limitations) one hypothesis in every possible time signature; it does not, however, include more than one representative of a given time-signature. That choice of that representative is determined by the offset-parent chooser, which is discussed in a later chapter.

Extending Hypotheses

As the startup module corresponds to that part of human audition responsible for making an initial guess as to the rhythm of a performance, so the *extender* corresponds to the processes which try to interpret the rest of the performance according to this guess. This chapter discusses the processes that extend hypotheses.

The Canonical Beat-Level Extension Problem

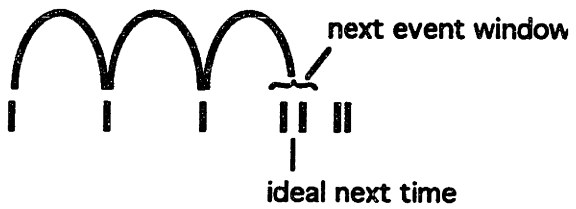


Figure Ext-1.

The figure above shows what we might term the "canonical beat-level extension problem." In the situation we are illustrating, it has been determined that the events to the left of the bracket are part of the level, and the system is about to consider new, incoming events, and determine which of them corresponds to the next beat. Events that will be considered as possible next beats are those which fall inside the *next-event window*, shown by the bracket. The width and location of this window depend on the times of the previous events that make up the level. The width of the window must be greater, as a percentage of Δt, as Δt gets smaller. The calculation of window width is shown in the graph in figure Ext-2.

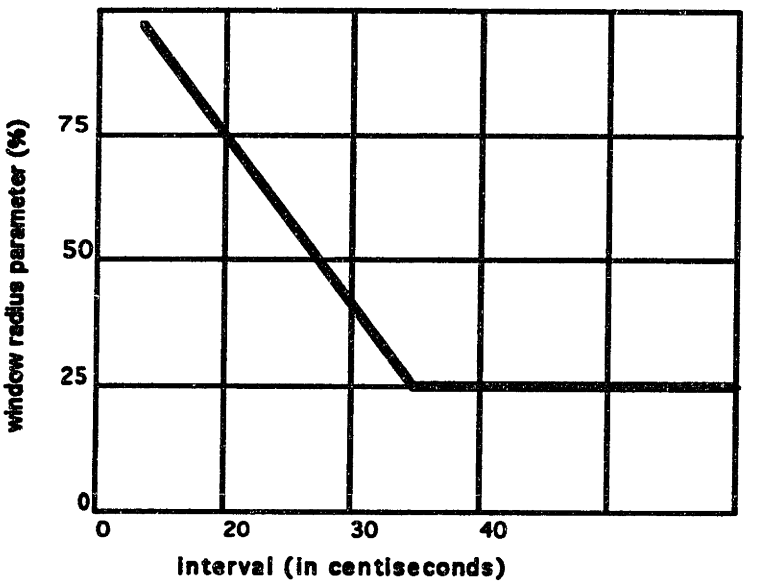


Figure Ext-2. Graph showing calculation of window radius parameter. The window radius parameter is the radius of the window expressed as a percentage of the interval. The graph shows that for intervals above 35 centiseconds, the window radius will always be 25% of the interval. For smaller intervals, the window radius will be greater as a percentage of the interval; at 20, for example, the window radius parameter is 75%.

The graph in figure Ext-2 reflects the conjecture that listeners tolerate more temporal sloppiness at lower (i.e., smaller-period) rhythmic levels than at higher ones. This effect is illustrated in figure Ext-3.

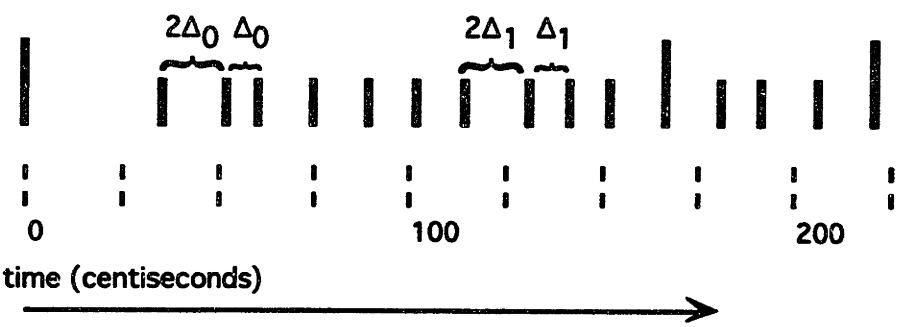


Figure Ext-3.

The vertical lines denote onset times from an actual performance; we see that some of the IOI's are actually twice as long as others. Aurally, this passage sounds sloppily played but nevertheless easily parsable as a run of sixteenths. At slower tempos, such a deviation would be unacceptable; even in passages characterized by heavy rubato, we do not normally find adjacent quarter-notes varying by factor of two.

Extension Cases

Having chosen our search window, the extension problem divides into three cases:

One -Event Case:

In the simplest case, there is exactly one event in the next-event window, and that event becomes the next beat:

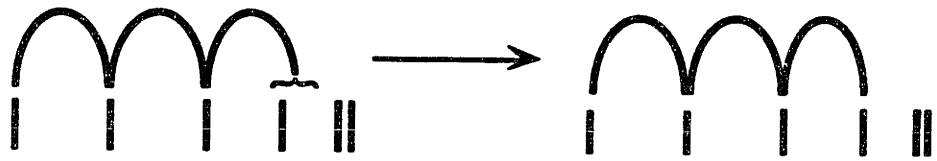


Figure Ext-4.

No-Event Case:

It may also happen that there are *no* beats in the window. In this case, we create a ghost event:

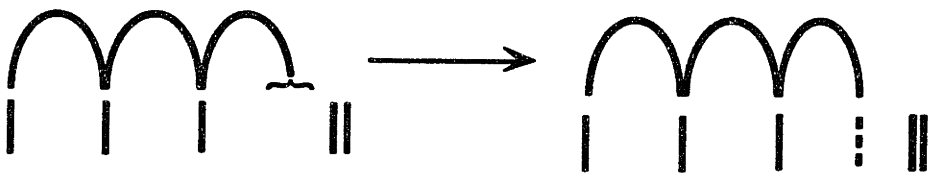


Figure Ext-5.

Several-Events Case:

In general, there will be several events in the window, and we must choose among them. Before we consider how this is accomplished, we must complicate our picture somewhat:

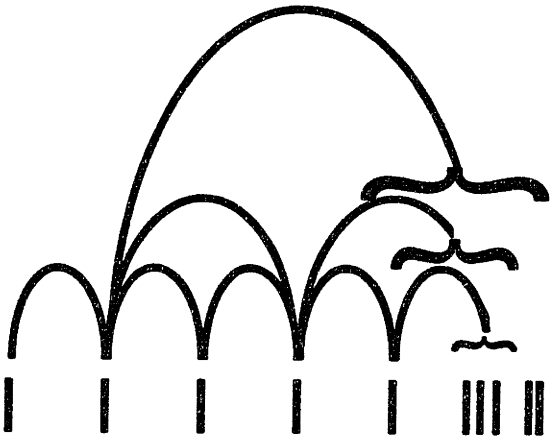


Figure Ext-6.

The situation we have been considering is a simplified version of the situation that we usually must consider, because a rhythmic hypothesis generally consists of more than one rhythmic level. When we extend hypotheses, we must consider all of these levels simultaneously.

In figure Ext-6 we see that the next beat at the eight-note level is also the next beat at the quarter-note and half-note levels, and that these levels must all agree on which event will be the next beat. The higher levels have their own next-beat windows. The candidates that we must consider are the union of candidates in any of these windows.

The candidates are passed to a module called *choose*, which can be thought of as a filter which eliminates unlikely candidates from consideration. (The choose module is described in detail in the next chapter.) If choose eliminates all but one candidate, we simply extend the hypothesis onto that event, and we are done with this step of the extension process. If choose returns more than one candidate, we create a new hypothesis for each candidate. This process of making several new hypotheses from one old one is called *splitting*.

Recall that the startup process produced a set of hypotheses with unique time signatures, called progenitors. The hypotheses that split from a single progenitor (or the extension of that progenitor) are called a *family*. All of the members of a family have the same time signature and different dispositions.

Alternative Subdivision

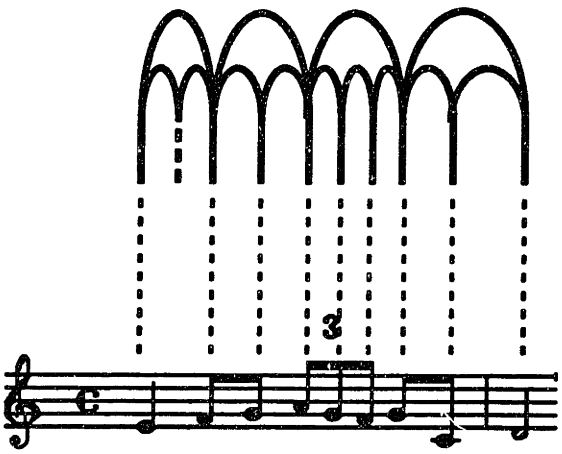


Figure Ext-7.

The alternative subdivision module is Machine Rhythm's way of detecting and parsing *non-isomerisms*, that is, changes in the way a level is subdivided. The overall strategy is straightforward. Whenever a new beat-node in the level which is immediately above the lowest level is constructed, we check to see if an alternative subdivision of that beat-node is possible. If the beat-node has two children, we try to construct a beat-node with three children; if it has three children, we try to construct a beat-node with two children. The routines that create the alternative subdivisions are more conservative than the routines that extend hypotheses in the normal way; we do not, for example, allow a new alternative subdivision to point to a ghost event, and we have tighter standards for temporal accuracy. So, for example, if we are changing from duplet subdivisions to triplets, all three constituents of the triplet must be actually present, and they must be accurately timed.

When the alternative subdivision is created, it is added to the list of hypotheses as a new progenitor, and it competes in the ranking process with the other hypotheses.

The restriction of alternative subdivisions to the level immediately above the lowest level helps us to avoid difficult logistical and performance problems, while accounting for most of the common non-isomerisms in Western music. There are, however, non-isomerisms that the system cannot parse because they occur at levels where we are not looking for them. Figure Ext-8, showing what is usually called a *hemiola*, is an example of such a non-isomerism.

Figure Ext-8. Example of hemiola from J. S. Bach, French Suite #6. Although the meter of the piece is (6 8), the second to last measure is parsed as (3 4).

Summary

The extender extends the rhythmic hypotheses produced by the startup process to include new events. The extension must be done in a manner that preserves the alignment of all the levels of the hypothesis. It may happen that two or more candidate events are acceptable candidates for the role of next beat; in this case, the hypothesis splits, and we have one new candidate for each accepted new beat. In this case we say that the hypotheses have different dispositions, that is, the way the arches subdivide is the same, but the events that they point to are different. Each new hypothesis is then maintained and extended just as the old one was.

By trying to keep track of several rhythmic levels at once, we vastly increase the bookkeeping required by the system. In exchange for this additional trouble, we obtain a benefit: the different levels reinforce and inform each other. When several different levels agree on a single choice for the next event, we can be fairly sure that that choice is correct. When they disagree, more computational resources should be devoted to making the decision, which is what happens when a hypothesis splits.

Choosing and Ranking

Since Machine Rhythm works by constructing a number of alternative rhythmic hypotheses, we need ways of deciding when some of those hypotheses are better than others. In this chapter we discuss strategies for choosing the best hypothesis from several alternatives, and for ranking hypotheses according to the likelihood that they match a human's interpretation.

Rhythm Tracking by Beam Search

Our overall strategy for tracking the rhythm is essentially beam search (Winston 1984). Beam search is a strategy for searching a tree of nodes[1], which is useful when there is an upper limit to the number of nodes that we can afford to examine, and a way of preferring some nodes to others. It works as follows:

1. Form a queue consisting of the root node.

2. Expand and collect all the child-nodes of every node in the queue.

3. Discard all except the N best nodes from those found in step 2, where N is the upper limit to the number of nodes we can afford to examine at each step.

4. Replace the contents of the queue with the nodes that remain after step 3, and go to step 2.

In our case, the nodes in question are hypotheses, and step 2, "expanding the child nodes" corresponds to extending hypotheses, as described in the chapter "Extending Hypotheses."

Machine Rhythm's hypothesis-finding strategy differs slightly from the "pure" beam search algorithm described above in that the processes that correspond to step 3 are opportunistic, in the sense that they will sometimes discard hypotheses even though they have not yet run into the hard limit of N. The reason for doing this is that some of the selection heuristics are more computationally expensive than others; we can sometimes conserve computational resources by discarding a hypothesis according to an inexpensive heuristic, thus preventing it from being processed by an expensive one.

---

[1]These nodes refered to should not be confused with *beat-nodes*. Descriptions of search algorithms usually describe search as taking place in a tree of nodes, leading to an unfortunate possibility of confusion in our case.

The routines that constitute step 3 are called *filters* and *rankers;* together they constitute the bulk of the computation carried out by Machine Rhythm. Filters and rankers take as input a list of objects, one of which is "right" in the sense that it is part of the rhythmic hypothesis that humans would prefer. The object in question may be the right event to form the next beat of a level, or the right parent level for a tactus, or the right hypothesis among several candidate hypotheses.

In some cases, we want a filter which returns only one answer. This will be the case if the consequences of getting the wrong answer are not fatal, as with the offset-parent chooser used in the startup phase. The event-chooser, discussed below, can take an argument which insures that it return only one answer. Usually we can accept more than one answer, but we want to discard answers that are obviously wrong.

In still other cases, we are not discarding any of the objects under consideration, but we need an order of preference established on them, because, when computational resources run low, we need to know which ones to discard. This is the case with the ranker.

All of these processes try to balance conflicting requirements of accuracy and efficiency. If they equivocate — return more then one answer, refrain from discarding a candidate — they run less risk of discarding the right answer, but they add to the computational burden on the system. If they discard candidates more readily, they run more risk of discarding the correct one.

Before we discuss the detailed workings of the filtering and ranking modules, however, we will consider some general issues involved in choosing or ranking objects. It is typically the case that if a filtering or ranking problem is non-trivial, it is because there are a number of different criteria that must be taken into account. Since multiple factors usually play a role in determining a choice of perceptual interpretation, we must almost always address this problem when building a perceptual model. The general considerations in this chapter apply not only to the problem at hand, but also, with appropriate alterations, to other perceptual modeling problems in machine audition and machine vision. (Minsky 1986, Jepson 1991).

Our approach is to formulate the problem as one of coordination of a number of specialists. Each specialist views the objects (here, events or hypotheses) from the point of view of its specialty. In Machine Rhythm, for example, a *timing* specialist prefers objects whose timing conforms to its notion of how beats should be timed, while a *melody* specialist embodies notions of how melodic patterns should relate to rhythm; a *density* specialist prefers events involving more notes (such as a chord, as opposed to a single note) as candidates for strong beats.

How do we manage specialists? If they all agree with each other, there is no management problem; we simply use their agreed-upon result. In general they will not agree, and must use various strategies for deciding which answer is correct:

1. Some specialists are good at making particular distinctions but not others. The timing specialist, as we will see, is good at telling us which of several (4 4) interpretations is correct, but is often useless for choosing between (3 4) and (4 4).

2. Some specialists work well in conjunction with other specialists. When the timing and density specialists agree, we can be fairly sure that they are both right.

3. Specialists differ not only in overall reliability but also in their ability to avoid false positives. The *melodic-pattern* specialist often produces no result at all, but when it does have a preference, it is relatively reliable.

4. When we can't do anything else, we can simply vote. The voting can be weighted, according to how much we trust one specialist over the others.

The way in which we combine results from the specialists also depends on the kind of answer we require. If our goal is to filter out inappropriate objects from a list, we can accept any object that *some* specialist determines to be optimal, and reject the others. If we need an overall ranking for the objects, we need a strategy for combining individual rankings established by specialists.

We will now consider the details of several of the filtering and ranking modules used in Machine Rhythm.

The Offset-Parent Chooser

During the startup procedure, we needed a means of choosing the best-offset parent for the tactus level (see chapter "Startup."), and we deferred to this chapter the discussion of how this is accomplished.

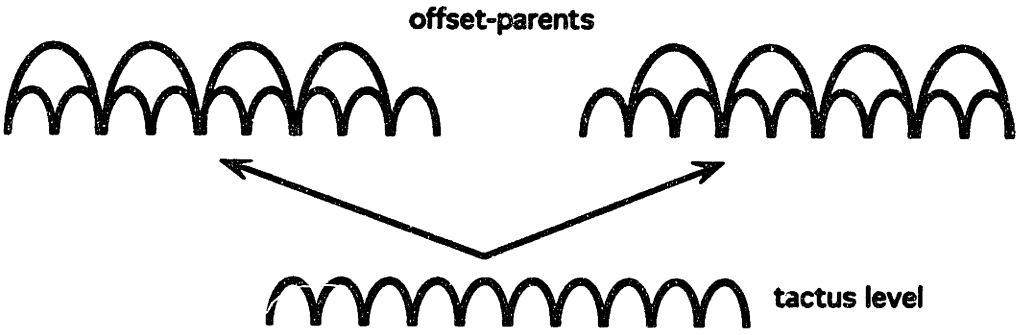


Figure Cho-1.

The parent beat-nodes should fall on the more salient events. Event salience, in turn, is determined by several factors:

1. Events are more salient if they are longer — both in the sense of ioi (i.e. the time to the next note) and duration (the time between note onset and release).

2. Chords are more salient than single events; chords with more constituents are more salient than chords with fewer, and chords that are more widely spaced — that is, chords with greater pitch-difference between the highest and lowest notes — will be more salient than chords which are less widely spaced.

3. Events that are preceded by a run of shorter events are perceived as more salient.

4. Actual events are more salient than ghost events.

The winning parent is determined by assigning each parent a score according to criteria 1-4, ranking the parents according to each score, and combining the ranks according to the weighted ranking scheme described above.

The Event-chooser Module

The *event-chooser* module is responsible for choosing among several candidates for the next beat. This module references a parameter called *comparison-functions*, which contains a list of functions that can compare two events according to some criterion for determining which event is more likely to be the next beat. The current list of comparison functions is

'(compare-chords compare-off-times compare-short-long-ness).

Each of these functions considers two events and decides which is the better candidate, or if the two events are equal with respect to the criterion in question. Compare-chords compares the density of the two events. If one is a single note, and the other is a chord, it will prefer the chord; if both events are chords, it prefers one event over another if it contains more notes, or if the range between the top and bottom notes is greater. Compare-off-times prefers events which are more *on time*, that is, closer to the time at which the hypothesis predicts the next beat will occur. Compare-short-long-ness prefers events which are immediately preceded by one or more notes with relatively short ioi's, as in figure Cho-2. These particular comparison methods are chosen because they are the fastest methods that we have for comparing events. We also have more sophisticated, costlier means of making these kinds of decisions, but we save those for use in the rank module, whose judgements, are more final. By making this list a parameter, we can easily revise our choice of event-comparing functions.

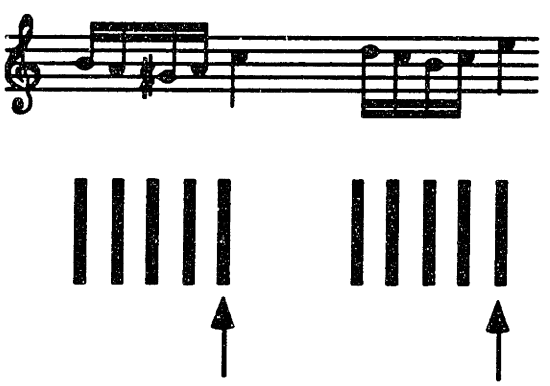


Figure Cho-2.

The choose module takes a list of candidate events and selects those that are the best according to at least one criteria. We'll call this stage the "primary election." These candidates then compete in a "run-off election," where a candidate gets a vote whenever it is the best according to a criterion. The module then returns the winner (or winners, in case of ties).[2]

If the choose module returns a single event, we simply extend the hypothesis onto that event:

[2]Note that this procedure is not equivalent to simply holding a general election and selecting the winner. In the case of a general election, it would be possible for an event which was not the preferred event of any specialist to win. The primary assures that the event chosen is also the preferred event of at least one specialist. Is this a better strategy? It's easy to think of arguments for and against, and the fact is, I really don't know. This is not a unique case; the structure of Machine Rhythm necessarily reflects many educated guesses and arbitrary choices.

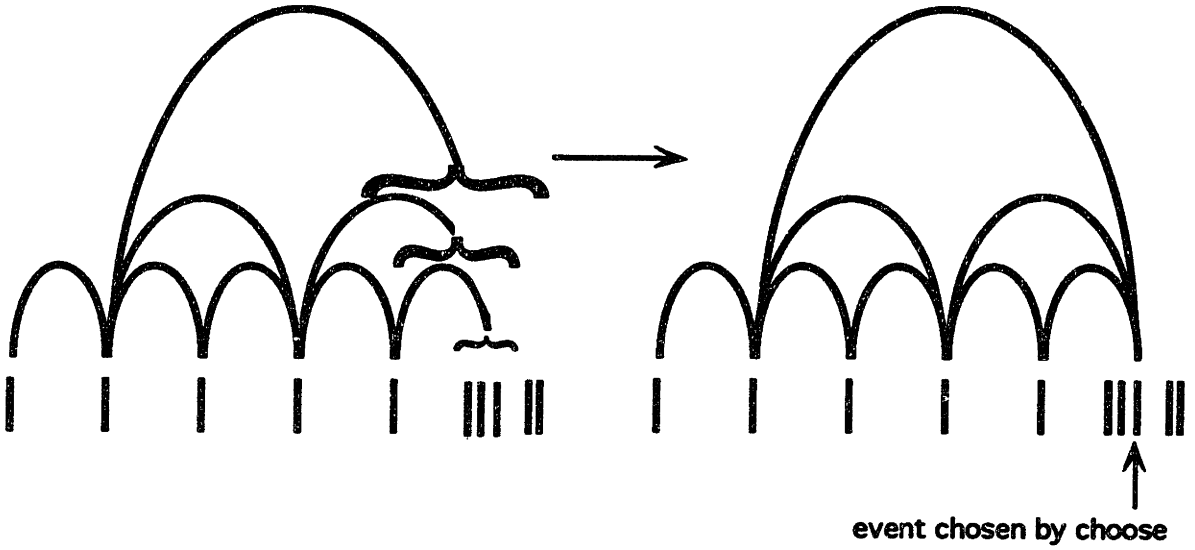


Figure Cho-3.

If the choose module returns more than one event, the hypothesis splits, as described in the chapter "Extending Hypotheses."

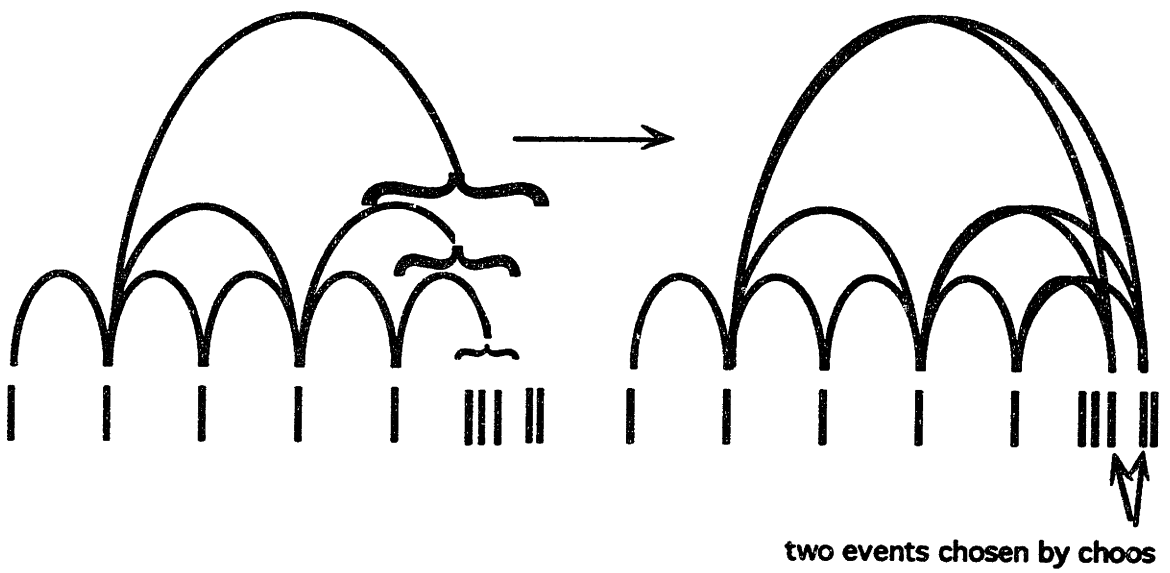


Figure Cho-4.

The reader may note that the choose module is not logically necessary for the operation of the program. We could simply generate a new hypothesis for every candidate event, and let the rank module decide which hypothesis is best. The reason we have the choose module (and in fact, all of the filtering routines) is that we want to keep the computational load on the rank module as light as possible. The hypothesis ranking process is the most time-consuming part of Machine

Rhythm; for this reason we want to avoid splitting hypotheses whenever we can, and when we do split, to produce the fewest possible new hypotheses. The choose module facilitates this by choosing the best candidate if the choice is obvious, and failing that, eliminating obviously unlikely choices.

Ranking and Pruning

The extension process described in the last chapter produces new hypotheses at a rate which is exponential with the number of events considered. Each hypothesis takes up part of the system's computational resources; it doesn't take long for the system to be swamped with new hypotheses. When the number of hypotheses, exceeds a certain threshold, the system ranks the hypotheses, and discards as many low-ranked hypotheses as necessary to bring the total down to a manageable number.

The rank and choose modules are similar in that they both have the responsibility for choosing among several candidates — the candidates being events in the case of the choose module, and hypotheses in the case of the rank module. The strategy in both cases is also similar; we allow a number of *specialists* to examine the candidates, and then we manage the results of the specialists' decisions. The specialists and the management scheme are both fairly simple in the case of choose; in the case of rank, both are more sophisticated. An important consideration in both cases is modularity of the management and the specialist functions; this allows us to add new specialists with minimal change to the management scheme, or alternatively, to change the management scheme without changing the way the specialists work.

The rank module differs from the event-chooser and the other filters in that it is the "court of last appeal" for hypotheses in machine rhythm; it is the final arbiter of which hypothesis is correct, and hypotheses that it ranks too low will be discarded by the pruner. For this reason, the rank module is the most conservative in making its decisions, and is given the most sophisticated (and computationally expensive) means of making judgments.

This conservatism allows the system to "err gracefully," and to recover from errors. As the system processes a rhythmically complex passage, it may happen that at various points the correct hypothesis is not the one which is ranked highest by the rank module. Hopefully, however, the correct interpretation will not receive such a low ranking that it is pruned away. It often happens that incorrect hypotheses attain a temporary advantage over the correct one. The incorrect hypotheses will usually get into trouble further on, and the correct hypothesis can again rise to the top.

The following series of figures show the processing of the troublesome passage of sixteenth notes that we encountered in the chapter "Introduction." They are difficult to process, first of all, because they are rather sloppily played; also, there is a spurious chord, the result of a performance mistake, which temporarily confuses the parsing process. During the processing of this passage, the correct hypothesis slips from first-place ranking, especially in the region surrounding the spurious chord; at the end of the passage, the correct hypothesis recovers its first place position.

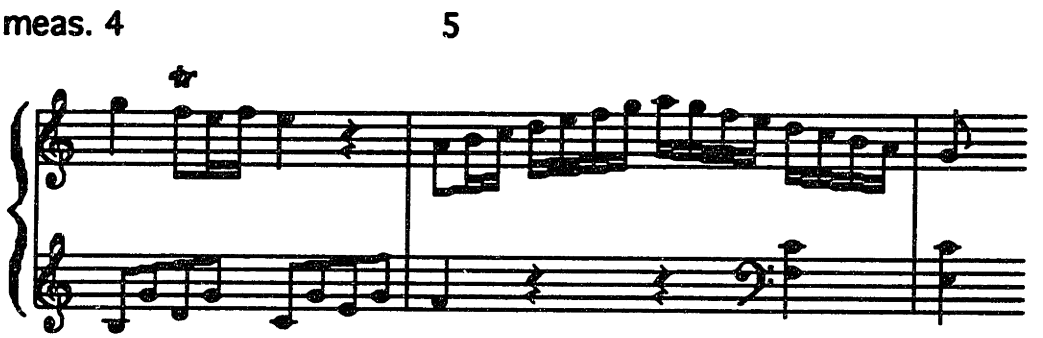


Figure Cho-5.

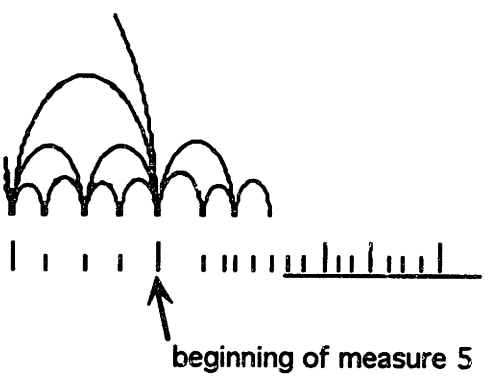


Figure Cho-6. Stage 1. At the beginning of the sloppy sixteenths passage, the correct hypothesis is ranked first. Each small arch represents an eighth note, so the small arches will point to every other event in the correct interpretation. The longer vertical lines represent onsets of chords, and the shorter vertical lines represent onsets of single notes.

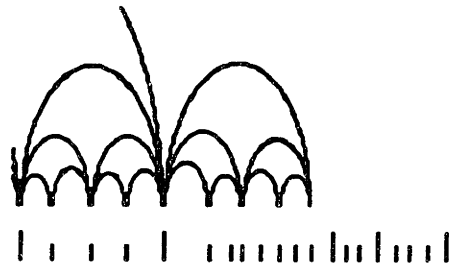


Figure Cho-7. Stage 2. After a few more events are processed, the correct hypothesis, shown above, has slipped to 6th place.

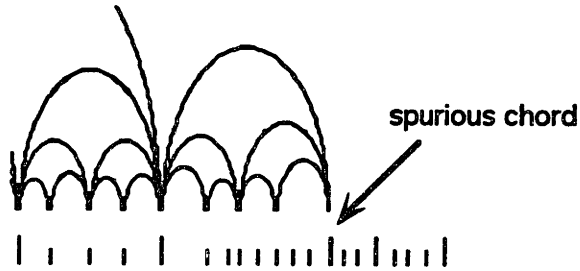


Figure Cho-8. Stage 2. At the same time, this is the top ranked hypothesis. Its rightmost arches have been pulled one event to the right by the spurious chord, caused by a performance mistake.

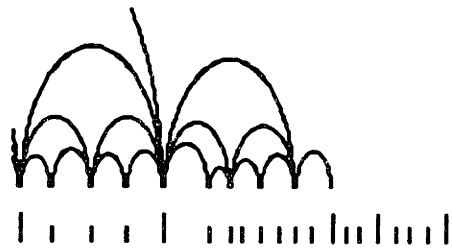


Figure Cho-9. Stage 2. Another incorrect hypothesis. This hypothesis also received a relatively high ranking (4th) because one of its nodes points to the spurious chord.

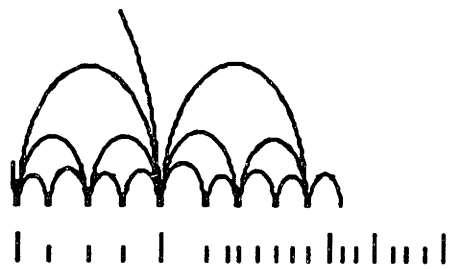


Figure Cho-10. Stage 3. The correct hypothesis, a little later. It has now slipped to 7th place, which is its worst ranking during the processing of this segment.

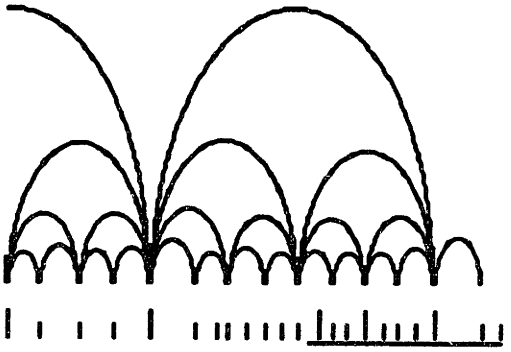


Figure Cho-11. Stage 4. Finally, Machine Rhythm finishes processing the passage, and the correct hypothesis, shown above, regains its number one ranking.

Like the choose process, the rank module has two main stages. First the hypotheses within each family are ranked, and then the highest ranking hypotheses from all the families are ranked against each other. (Recall that a family is a set of hypotheses descended from the same progenitor; family members have the same phase and subdivision scheme, but possibly different dispositions.)

Within-Family Ranking

1. *Density*

Density measures the number of notes that comprise a summary event, and how widely spaced they are; we have already encountered this concept in the chapter "Startup." A note is denser than a ghost-event, and a chord is denser than a single note; if two chords contain the same number of notes, the chord which is more widely spaced will be (according to our definition) denser.

The density of a hypothesis is $\sum_{e \in E} n(e)d(e)$, where E is the set of summary events pointed to by some node in the hypothesis, d(e) is the density of the event e, and n(e) is the level of the highest-level node which points to e (i.e., n(e) for lowest level nodes is 1, for next to lowest-level nodes is 2, etc.). Hypotheses which place higher-level nodes on denser events will have higher density scores. (This is the reason that the incorrect hypothesis was ranked highest in figure Cho-8, above.)

2. *Short-long*

Events are perceived as emphasized when they are preceded by a run of shorter events (Povel 1985). The short-long scorer counts the number of shorter events that immediately precede an event — here, the length of an event refers to its ioi, and we say that event A is shorter than event B if A's ioi is roughly half B's or less. An event's short-long score is 0 (no short predecessors), 1 (one short predecessor), or 2 (two or more short predecessors). The short-long score of a hypothesis is then calculated in a manner analogous to the hypothesis' density score: $\sum_{e \in E} n(e)s(e)$, where s(e) is the short-long score of event e, and the other terms are the same as in the calculation of the density score.

3. *Max-duration*

Another factor affecting the perceptual salience of events is their absolute lengths (as opposed to the relative lengths, measured by their short-long score). The max-duration of a single (non-chord) event is the maximum of its ioi and its duration (time between onset and offset). The max-duration of a summary event is the maximum of the max-durations of its constituents. The max-duration score of a hypothesis is calculated analogously to the calculation of the density and short-long scores.

4. *Timing*

Another way to compare hypotheses is to compare the accuracy of their timings — that is, in which set do successive beat-nodes have the most equal intervals. This method of comparing hypotheses would prefer, for example, the hypothesis in figure Cho-13 to that in figure Cho-12.

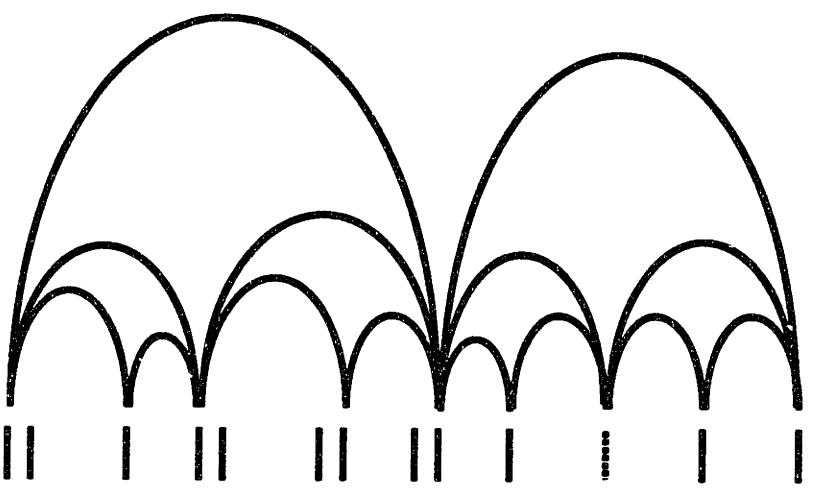


Figure Cho-12.

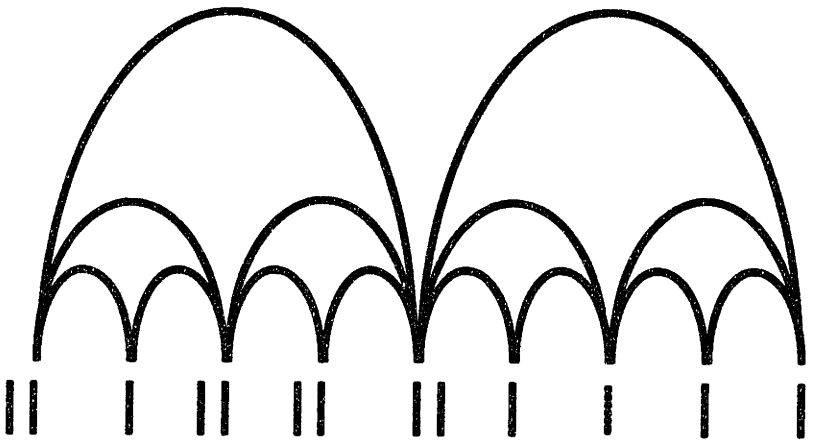


Figure Cho-13.

Note that this method would also tend to penalize the hypothesis in figure Cho-14.

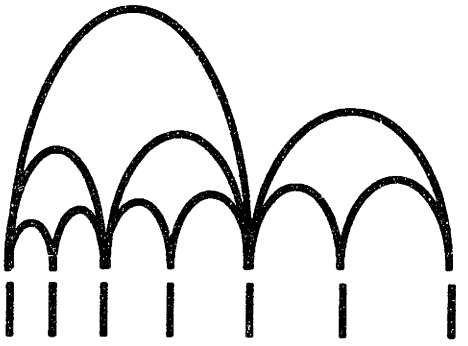


Figure Cho-14.

This is not exactly what we want, since the hypothesis of figure Cho-15 is actually a fairly reasonable one; it says, essentially, that the performer is playing a ritardando. Within each level, each arch gets progressively wider, and the change is regular and predictable. In other words, we don't want to consider the absolute change in interval size, but how irregular the change is. The timing score of a level, therefore, is the *standard deviation* of the change in interval size over the level. This system will reward hypotheses whose intervals either do not change much at all, such as the hypothesis shown in figure Cho-13, or else change in a constant manner, such as the one shown in figure Cho-14 but punish those that vary irregularly, such as the hypothesis shown in figure Cho-12. The score for a hypothesis is the score for the levels, summed over the levels in *levels-to-use-for-timing*, which (currently) consists of the lowest and next-to-lowest level. (With timing, low scorers, i.e. those whose intervals' derivatives have the lowest standard deviation, win.)

5. *Pattern-within-families*

Hypotheses can also be compared according to which are most consistent with repeating musical patterns. In the following example, hypothesis H is preferable to hypothesis H', because H includes the note which continues the established melodic pattern.

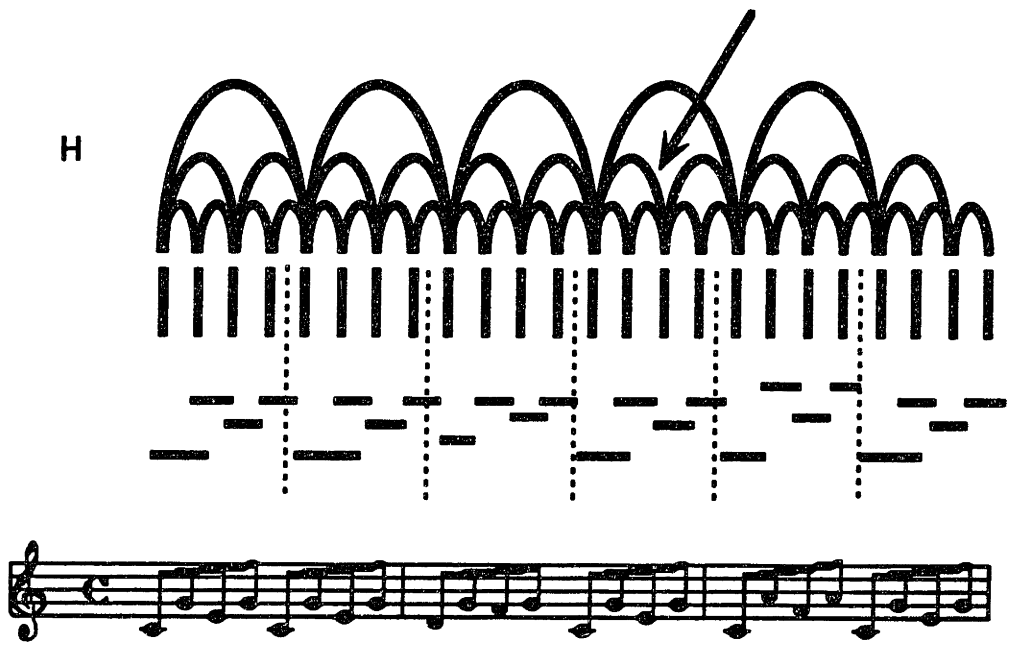


Figure Cho-15.

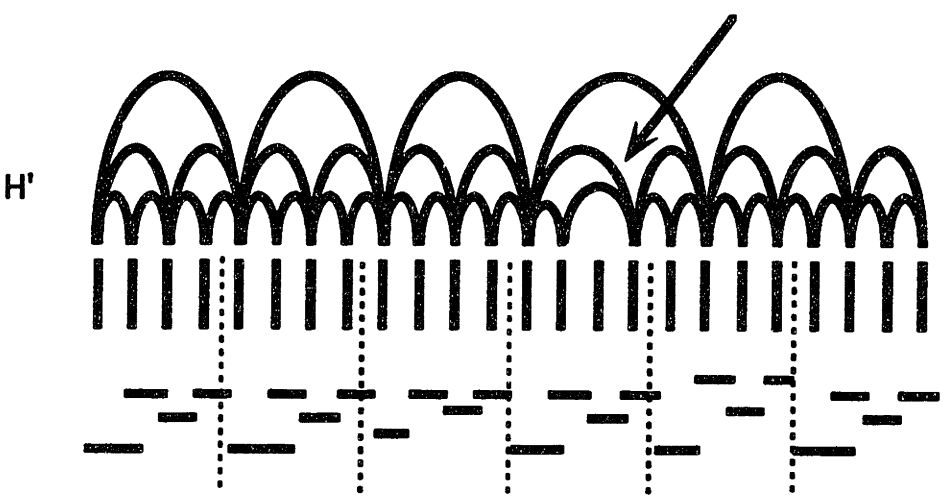


Figure Cho-16. These two hypotheses diverge during the fourth occurrence of the pattern, at the point indicated by the arrows. The first hypothesis (figure Cho-15) remains aligned with the pattern, while the second hypothesis (figure Cho-16) does not.

The procedures for extracting pattern information are discussed, appropriately enough, in the chapter "Extracting Pattern Information."

After the hypotheses have been scored and ranked according to these criteria, we have to select an overall winner for each family. First we look for "trustworthy conjunctions of specialists," that is, cases where certain pairs of ranking systems both ranked a single hypothesis in first place. If either pattern-within-families or timing decide that a certain hypothesis ranks first, and one of the other ranking methods agrees, we select that hypothesis, and eliminate the others in the family from consideration.

If no such conjunction of specialists pertains, we use the *weighted voting procedure:* Each hypothesis is assigned a score $\sum w_i r_i$, where the r_i range over the ranks for each criterion, and w_i is a measure of how much we trust the rank r_i.

Between-Family Ranking

After completing the within-family rankings, Machine Rhythm ranks the winners from each family. When ranking hypotheses from different families, we use a somewhat different set of criteria from those used to rank hypotheses from the same family. When we are ranking within families, the way that the nodes are grouped is fixed, and it is the disposition of the bottom-level nodes — which events they point to — that varies. When we are ranking between families, we are comparing different ways of grouping the bottom-level nodes.

1. Density, Short-long, and Max-duration

Density, short-long, and max-duration comparisons are used in the same way when comparing hypotheses between families as they are for comparing hypotheses within families.

2. Timing

The timing-ranker is essentially looking at the "evenness" of hypotheses. It is reasonable to assume that hypotheses from the same family will have different degrees of evenness, since the only way in which they differ from each other is to point to different events, like the hypotheses shown in figures Cho-17 and Cho-18

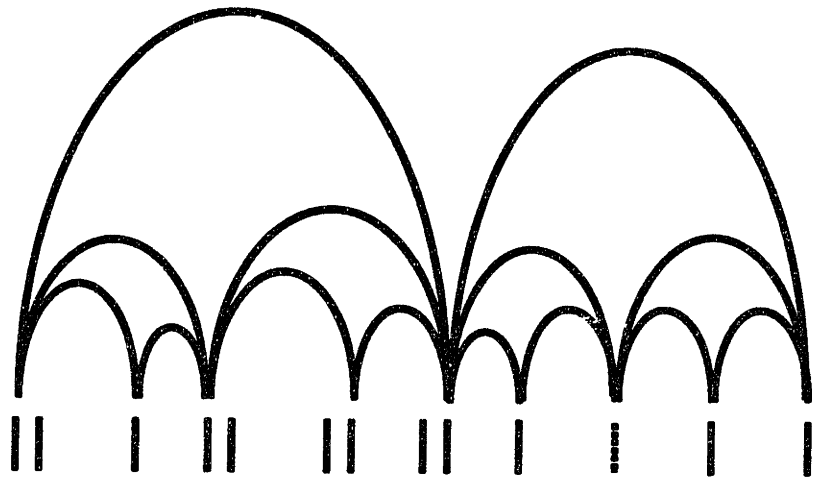


Figure Cho-17.

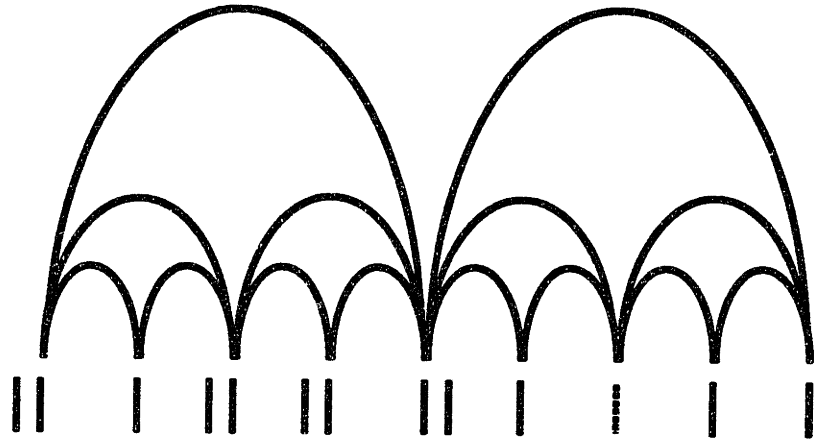


Figure Cho-18.

The winning hypotheses from separate families, however, may well point to the same events:

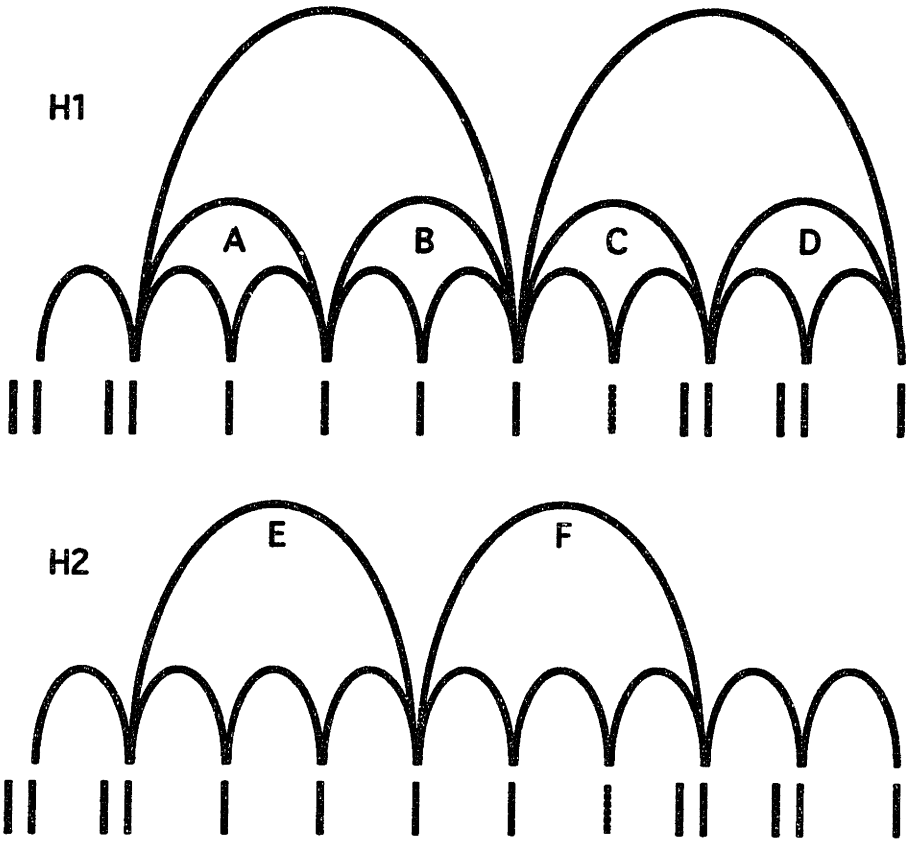


Figure Cho-19.

They may, however, have different timing scores, since the timing score is based on more than just the bottom level — so, the timing score for H1 in figure Cho-19 will be partly determined by the beat nodes labeled A, B, C, and D, while the score of H2 will be partly determined by the nodes labeled E and F. We would like for the two hypotheses in figure Cho-19 to be considered equivalent from the point of view of timing; we are not interested in the slight differences in timing-score that result from grouping the bottom-level nodes differently. Therefore, before we compare the timing scores of events from different families, we make sure that their bottom levels are actually disposed differently.

3. *Pattern-between -families*

The pattern analysis routines (discussed in the chapter "Extracting Pattern Information") produce a result called the *offset-suite* for each level. This is a list of offsets at which patterns repeat themselves. The offset suite for the lowest level of hypothesis H in figure Cho-15 is (2 4 8),

meaning that there is a pattern repeating after two notes (the basic up-down alternation), at four notes (the pattern C-E-G-E) and eight notes (twice the four-note pattern).

The pattern-between-families procedure uses the offset-suite to scores hypotheses on how well they are confirmed the pattern information. Suppose, for example, that the lowest level of the hypotheses shown in figure Cho-19 has offset-suite (2 4 8). This offset suite will confirm the second and third levels of hypothesis H1, but none of the levels of H2.

Pattern-between-families and pattern-within-families are examples of ranking procedures that have limited domains. Pattern-between-families is useful for determining which of several alternative time-signatures — that is, ways of subdividing beat-nodes — is correct. This means that it is useless for judging between hypotheses from the same family, since, by definition, they all have the same time signature. Pattern-within-families, on the other hand, looks at how closely the events that a hypothesis points to follow the predictions of a found pattern. This is useful when the hypotheses being judged tend to point to different events, as is the case for events in the same family, but not when the pointed-to events are the same, as is often the case for the winning hypotheses from different families.

After the winners from each family have been chosen, we combine the separate ranks into an overall ranking with the weighted voting procedure.

Use of Loudness Information

It may seem somewhat odd that there is no "loudness expert," that is, there is no greater importance assigned to notes with higher MIDI velocity. The reason for this is that we failed to find, in any of the available MIDI data, any indication that performers were using MIDI velocity to indicate downbeats. This is somewhat contrary to what our intuition would lead us to expect, and there is experimental evidence that listeners will in fact use loudness information to form rhythmic parsings if it is available (Norris 1990). It would appear, however, that performers only use velocity to indicate meter when no other cues are available (Palmer 1990). A loudness expert would not be difficult to construct if it were determined that one would be helpful.

Pruning

When the number of hypotheses exceeds the *initiate-pruning-threshold*, we have to prune away the less likely candidates. When we are pruning, we make the conservative assumption that the correct hypothesis is not ranked first, and we must avoid discarding it.

If the correct hypothesis is not ranked first, where is it? It may be that the wrong family is ranked highest, and that the correct hypothesis is the winner for another family, or it may be that the correct hypothesis was not chosen as the winner in its family. We therefore maintain two parameters, the *maximum-in-each-family*, and the *maximum-number-of-families*; their product is *maximum-hypotheses-after-pruning*. Whenever we prune hypotheses, we retain only *maximum-in-each-family* hypotheses in each family, and *maximum-number-of-families* total families.

Machine Rhythm's accuracy and performance depends critically on these parameters. When they are small, the program runs faster, but the chance that the correct hypothesis may be pruned away is higher. When they are large, there is less chance of discarding a good hypothesis, but the system runs more slowly.

Extracting Pattern Information

This chapter discusses Machine Rhythm's strategies for detecting and using information provided by melodic patterns.

Music often contains repeating patterns; these patterns may occur in the melody or the accompaniment, or in both, as in figure Ext-1. In fact, it doesn't make much sense to talk about music being composed of themes or motives, unless variants of the theme or motive in question occur more than once. A motive is often repeated, either exactly or in variant form, immediately after being introduced, and the time between occurrences of the motive is usually a rhythmically important interval.

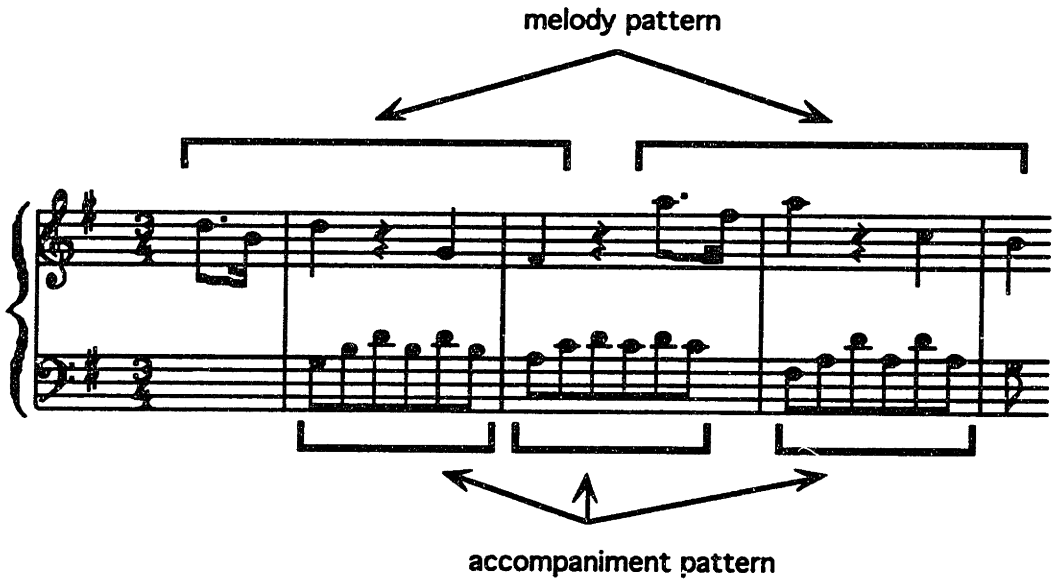


Figure Ext-1

How can we exploit patterns to determine the music's rhythm? One way is to prefer hypotheses whose lowest levels are consistent with the prediction implied by a repeated pattern. Of the hypotheses H and H' below, H is preferable by this criterion, since note n_1 is predicted by the repeated pattern, while note n_2 is not.

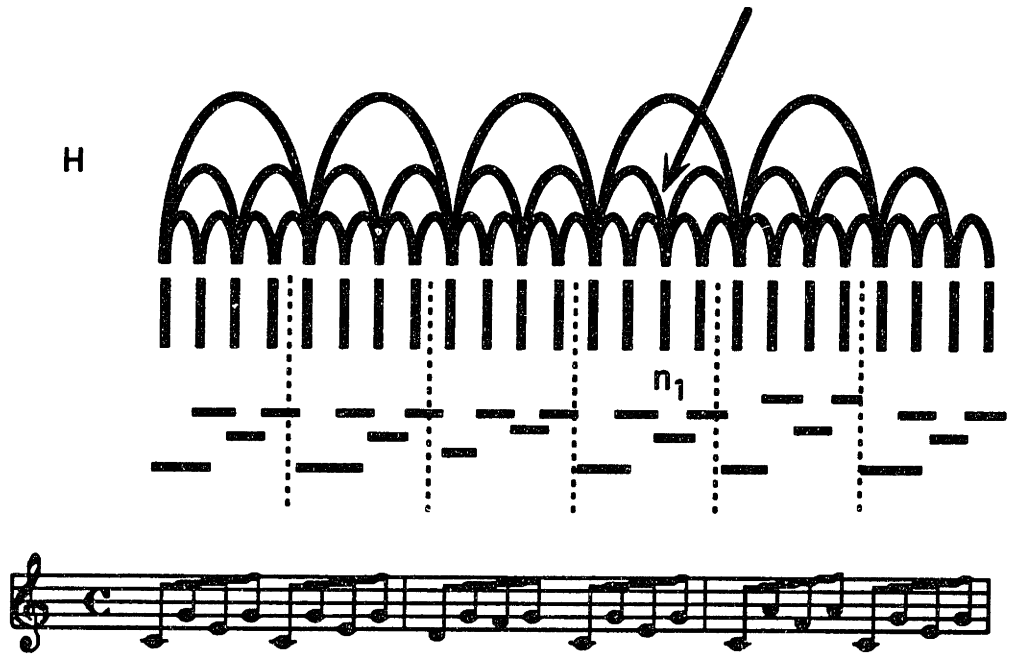


Figure Ext-2.

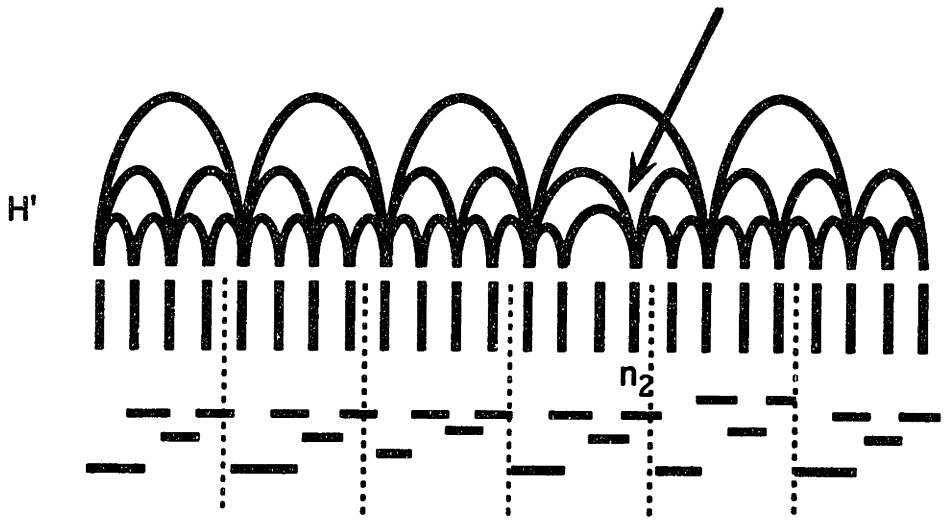


Figure Ext-3.

Another way is to use the time interval between occurrences of a pattern to confirm or disconfirm hypotheses. In figure Ext-4, hypothesis H is preferred to hypothesis H'. The third level (from the bottom) of H is confirmed by the pattern of length 4, since each third-level beat of H subsumes four eighth-notes. This is not the case for H', since each of its third-level beats subsumes six eighth notes. Third level beats of H' occur sometimes at the beginning, and sometimes in the

middle of the four-note pattern, while the beats of the third level of H are always at the beginning of the pattern.

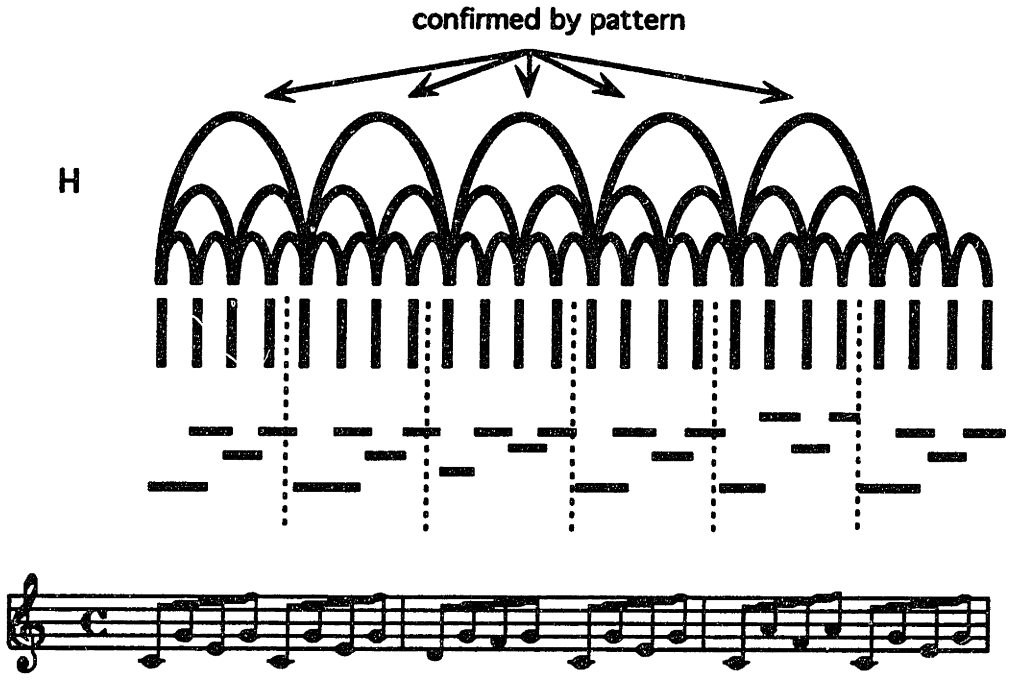


Figure Ext-4.

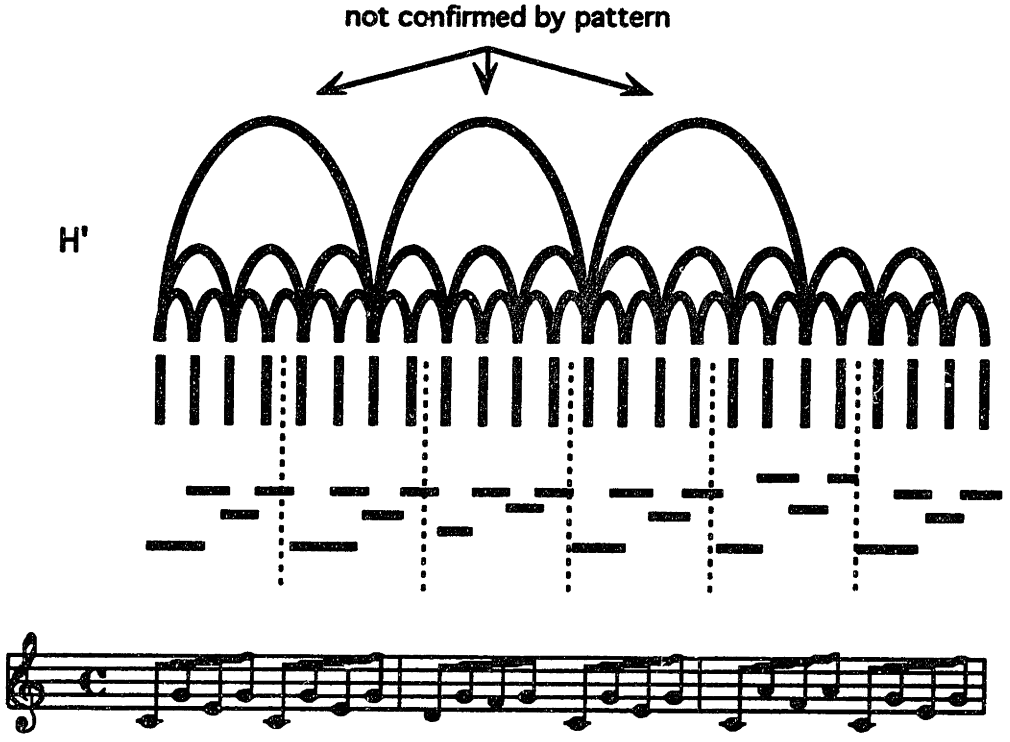


Figure Ext-5.

Pattern matching is the most computationally intensive process in Machine Rhythm's ensemble of ways to rank hypotheses; in general, the process takes time proportional to n^2k^2, where n is the number of summary-events to be matched and k is the number of constituent events in each summary event. If we also consider that we must try to allow for near misses, we see that the number of computations required can be quite large. For this reason we use a number of heuristics to prune the search for matches:

1. We note that for the passage in figure Ext-4, matches occurred at offsets of 2, 4 and 8. We will refer to such offsets as *matching offsets*. As we have pointed out, it is not a coincidence that these are the important rhythmic subdivisions of the piece — that is, eighth notes are grouped in twos to form quarter-notes, which are grouped in twos (that is, four eighth notes), to form half-measures; a measure will be eight eighth-notes in length. Since it is usually the case that matching offsets will be multiples of other matching offsets, we can avoid looking for matches at *every* offset. Once we find a matching offset, say, 2, we need only check the offsets 2s, where s is a permissible beat-node factor — in our case, either 2 or 3. In the case of figure Ext-4, s = 2 will be the better choice; we then check $2 \bullet 3s$, for $s \in \{2,3\}$. The sequence of numbers obtained in this way — in this case, (2, 4, 8) is called the *offset suite* of the level.

2. As we noted in the chapter "Preprocessing," the process of matching events can be made less computationally expensive by assuming that the matching notes for a given motive are in the same voice — in fact, that was our primary reason for separating the voices in the preprocessor.

When do we consider two motives to be variants of each other? Clearly it is not simply that their pitches match, as the example in figure Ext-2 shows. Nor, as we can also see from figure Ext-2, do the pitch-intervals need to match exactly; the second two occurrences of the four-note motive are C-G-E-G and D-G-F-G; the pitch intervals off successive notes (in semitones) are (7, -3, 3) and (5, -2 2).

The case can be made that we consider motives A and A' to be variants if their *contours* are the same — in other words, we could test whether A' is a variant of A by encoding them both as series of 1's (for rising pitch), -1's (for falling pitch) and 0's (when pitch stays the same), and comparing the encodings. But while comparing contours seems to be the right idea in general, it is easy to find cases where the method can yield false positives or false negatives. The following examples show two melodic contours that match themselves about equally well (as measured by percentage of matches) when offset by 6;, however, we would like the first to be considered a match but not the second.

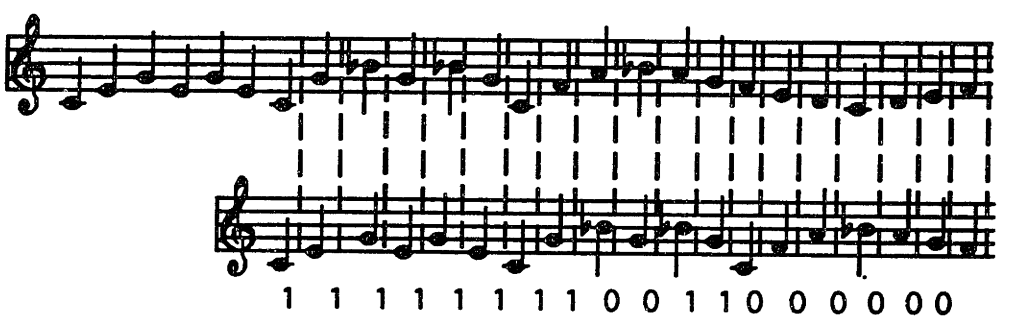


Figure Ext-6.

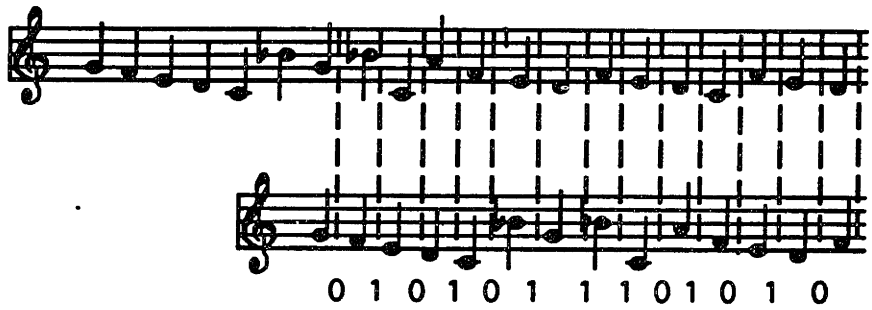


Figure Ext-7.

In the first case, we have a melody which contains a motivic contour; the contour is repeated twice, and we begin to hear it a third time, after which the melody does something different. As a result, about half the contours match when the melody is shifted by six notes and compared with itself. In the second melody, there is no six-note contour, and the 53% match score is due to chance, which is about what we would expect. What distinguishes the two cases is the *burstiness* of the matches. In the first example, the 1's almost all occur in a single burst at the beginning, since that's where the repeated motive is. In the second example, the 1's are more evenly distributed. The burstiness of matches is measured by comparing the frequency of intervals between 1's with what one would expect from a Poisson distribution, and calculating the mean-square difference.

The procedure for extracting information about patterns, then, can be described as follows:

1. Given a level, we derive a sequence of pitches A (by listing the pitches associated with each node in the level). We then calculate the pitch-sequences A_2 and A_3, which are A offset by 2 and 3, respectively. We first compare the pitches of A vs. A_2, and A vs. A_3; we allow pitches to match if they differ by no more than two semitones. The percentage of pitch-matches for each offset's match-attempt is its *pitch-score;* if one pitch match is significantly better than the other, we go to step 4.

2. Otherwise, we compare the contours of the melody and the offset melody; again, if one match is significantly better than the other, we go to step 4.

3. If neither steps 1 nor 2 determined which offset matches better, we check to see if the matches in one case are significantly burstier than the other.

4. If steps 1-3 concluded that one of the offsets, say k, produces a better match than the other, we add k to a list called the *offset-suite*. We then repeat steps 1-3 using offsets 2k and 3k in place of 2 and 3. If neither 2 nor 3 was judged the better offset, we continue to search for an offset that produces a match. (We would try 4 and 6 next.)

5. In addition to producing the offset-suite, we calculate the *best-offset* — the offset with the highest match-score. In the case of the pattern in figure Ext-6, the best-offset would be 6; in general the best offset should be the length of the "basic" repeated pattern.

Summary

Machine Rhythm's pattern-extraction routines produce two kinds of information about each level: one is the offset-suite, which is a list of offsets at which the melodic pattern of the level matches itself. The other is the best-offset, which is the offset of the basic melodic pattern of the level (if there is one). The information in the offset-suite is used to confirm or disconfirm hypotheses, since the subdivisions of beats in the correct hypothesis should match the offsets in the suite. The best-offset can be used to predict the next event for a level, based on the assumption that the pattern that has been extracted continues.

Context-Dependent Links

This chapter discusses context-dependent links, a knowledge-representation scheme used in the Machine Rhythm program.

In knowledge representation systems that deal with large numbers of objects, it's desirable to represent the objects as parsimoniously as possible. This is often accomplished by means of a *taxonomy* — a hierarchical network of nodes connected by "a kind of" links. Suppose, for example that we want to represent different kinds of animals. One way would be to simply create a frame for each animal that we want to represent, as in figure Cont-1.

TIGER	LION	LEOPARD
attributes:	attributes:	attributes:
Four-legged	Four-legged	Four-legged
Furry	Furry	Furry
Carnivorous	Carnivorous	Carnivorous
Striped	Has Mane	Spotted

Figure Cont-1.

The problem with this representation is that it is redundant; the attributes "four-legged," "furry," and "carnivorous" occur once in each description, and would be repeated for every other feline that we described. A better way is shown in figure Cont-2.

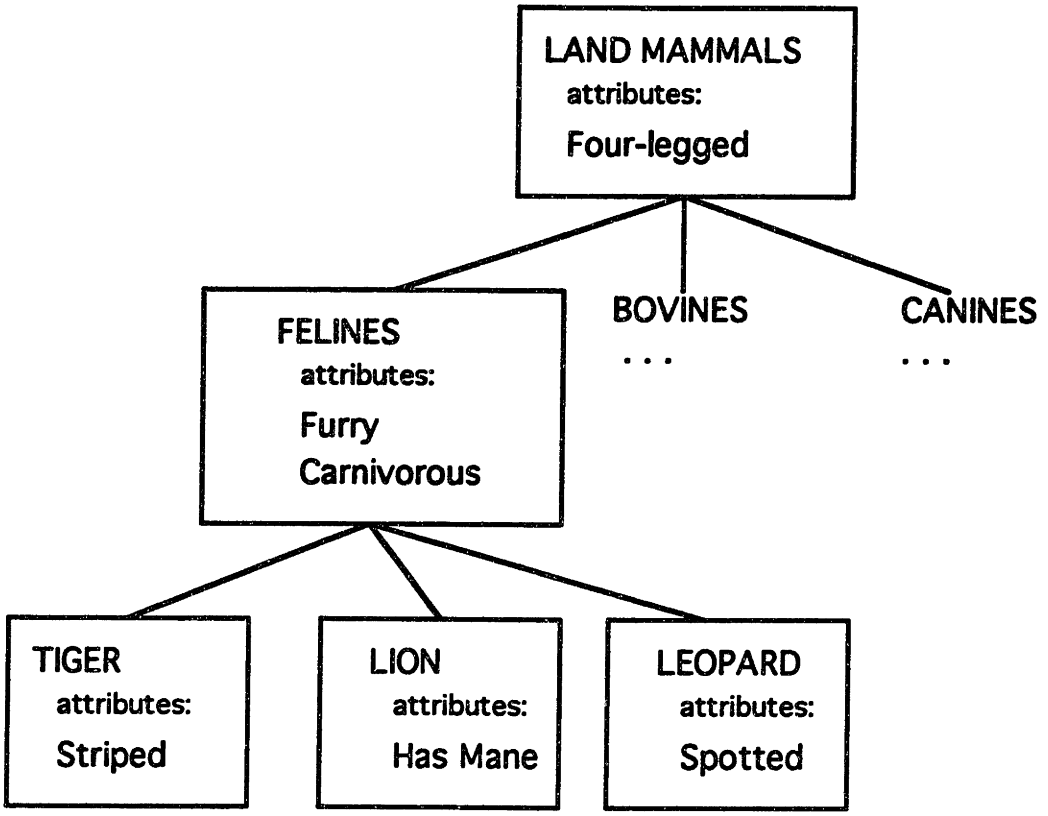


Figure Cont-2. Taxonomy of large cats.

Here we have nodes representing *kinds* of animals, such as felines, which are always furry and carnivorous. It is understood that each animal's attributes include all of the attributes in nodes which are above it in the hierarchy, in addition to its own. Thus a tiger is striped, because that is an attribute particular to tigers, but it is also furry and carnivorous, because those are attributes that pertain to all felines, and four-legged, since it is a land mammal. The properties of these kinds of hierarchies, their variants and associated problems have been treated extensively in the literature of knowledge representation. (Minsky 1981, Lenat 1990).

It may also be the case that we need to parsimoniously represent *networks* of nodes rather than individual nodes. Suppose we want to represent paths to various locations near my office at the Media Lab, as in figure Cont-3:

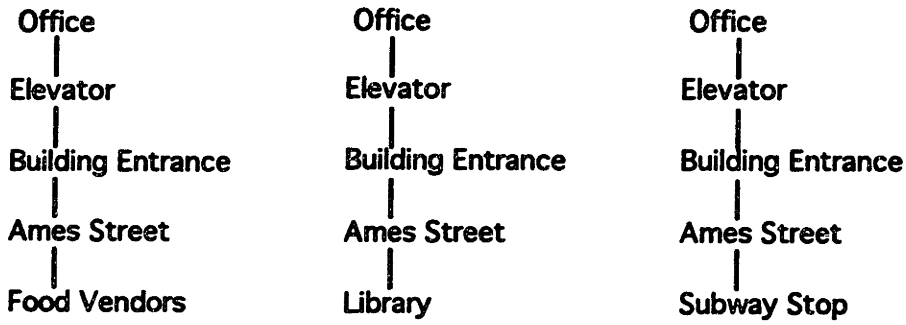


Figure Cont-3.

Here the nodes represent locations, and the interpretation of each link (which in figure Cont-2 was "a kind of," as in, "A tiger is a kind of feline,") is now "go to." (So, to to go the food vendors, I start in my office, go to the elevator, go to the building entrance, etc..) Like the representation in figure Cont-1. the representation in figure Cont-3 is redundant; most of the information is duplicated in each path. We could, instead, use a representation that is essentially a map of the local area, as in figure Cont-4.

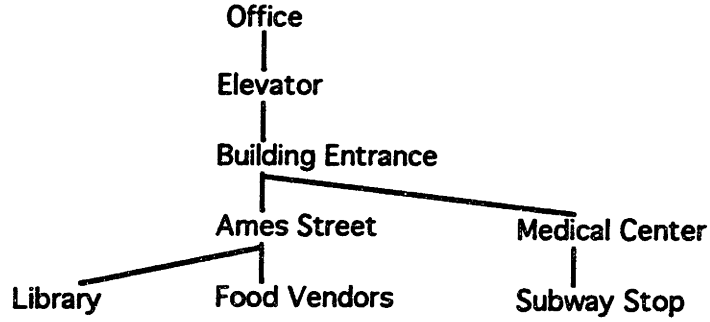


Figure Cont-4.

While this representation could be said to represent "all possible paths" to the food vendors, the library, or the subway stop, it doesn't represent any particular path, because it doesn't, for example, say where we should go from "building entrance" — whether to "Ames Street" or to "medical center." What we would like is a kind of magic map, which only shows the links that we are interested in for a certain purpose. If we wanted to go to the subway, for example, only the links that take us to the subway would be visible,

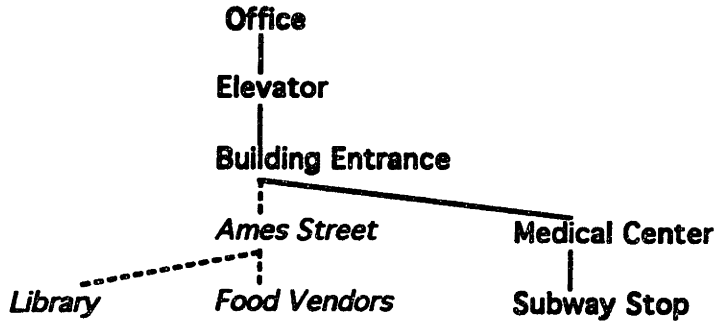


Figure Cont-5. The solid lines represent visible links, and the dotted lines represent invisible links.

whereas if we were interested in going to the food vendors, a different set of links would be visible:

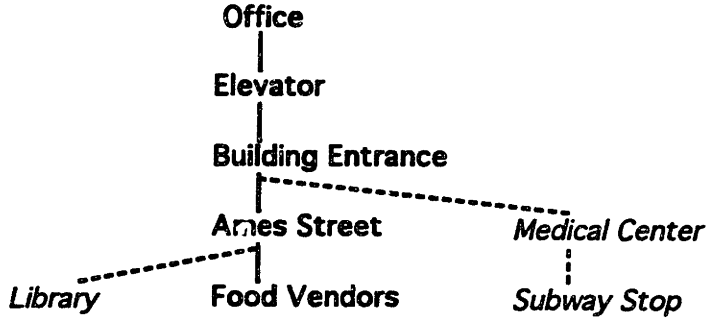


Figure Cont-6

The new representation of a path, then, is the map from figure Cont-4, together with a record of which links are visible in the context of that path. Links that work in this way will be called *context-dependent links.*[1]

In the implementation of context-dependent links used in Machine Rhythm, there is a global variable called *context*. The value of a context-dependent link depends on the value of this global. Suppose that we are at "building entrance," and *context* is set to :subway-path. When we query for the value of the context-dependent link leading down from "building entrance," the link determines that the value associated with :subway-path is "medical center." Had the value of *context* been :food-vendors, the query would have returned "Ames Street."

[1]This scheme was described and elegantly implemented in Lisp by Alan ("Lansky") Ruttenberg, whose paper on the subject I would happily cite if he would just write it. They are similar to K-lines (Minsky 1986) and to schemes used in NETL (Fahlman 1979).

A problem with this implementation of context-dependent links is that every time we add a new location node, all of the previously created links have to be updated. If, for example, we added a new link, say "reference section" below library, we would create a new context, :reference-section, which would control the appropriate link from "library." We would then have to inform the links from "building entrance" and "Ames Street" of the existence of this new context, so that they could be turned on at the appropriate time. We can overcome this problem by refining our notion of context. Instead of simple tokens, contexts can be structures which can inherit the properties of other contexts. So, for example, we start with the contexts :ames-street and :medical-center, which would determine which link leading from :building-entrance was visible:

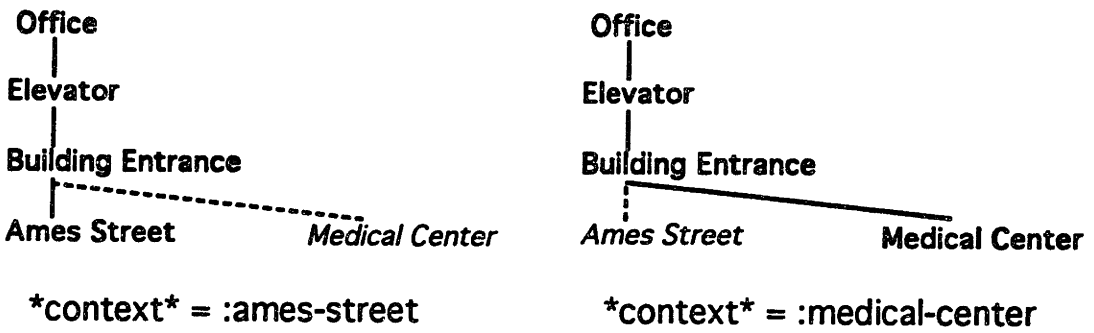


Figure Cont-7

We then create the context :food-vendors:

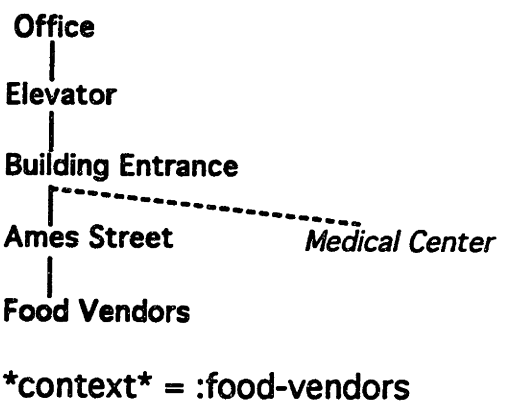


Figure Cont-8

The :food-vendors context inherits from the :ames-street context (we will also say, "the :ames-street context is an ancestor of the :food-vendors context," and "the :food-vendors context is a descendent of the :ames-street context."). When *context* is set to :food-vendors, and we query the link leading from "building entrance," it discovers that it has no value associated with :food-vendors. The link therefore checks to see if :food-vendors has any ancestors, and discovers that

:food-vendors inherits from the :ames-street context, which tells the link that its value should be "Ames Street." Similarly, the :library context inherits from :ames-street and the :subway-stop context inherits from :medical-center.

What is the difference between the context-dependent links scheme and the frames-with-taxonomy scheme? In terms of descriptive power, there is none, and we could, for example, represent paths as frames with inheritance, with the same degree of parsimony:

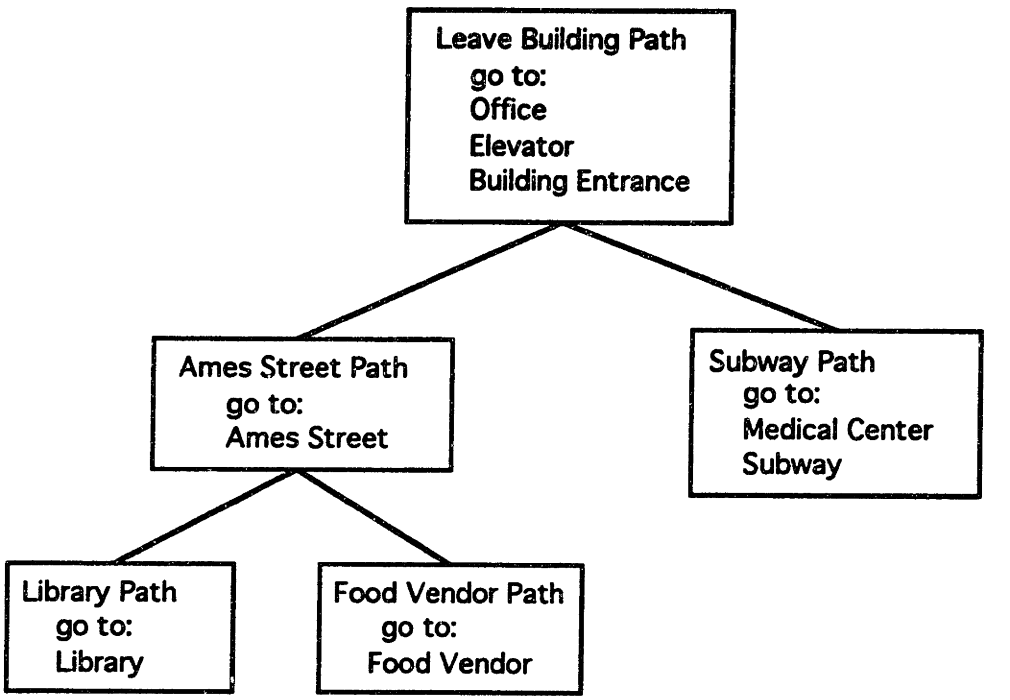


Figure Cont-9

Figure Cont-9 shows a "taxonomy of paths-frames;" here the links have the interpretation "before you do what is in this node, do what is in the node above." The difference between the two schemes is in where information is stored, and what we find it convenient to consider an "object." In the path-frames scheme, the objects are paths, and information about where to go is stored in the path-frames; in the context-dependent links scheme, the objects are locations, and information about where to go is stored in the links.

Using Context-Dependent Links to Represent Hypotheses

Context-dependent links provide a natural solution to problems that arise when we need to represent multiple hypotheses in Machine Rhythm. Suppose that we want to represent a (3 4) hypothesis and a (4 4) hypothesis, as shown in figure Cont-10:

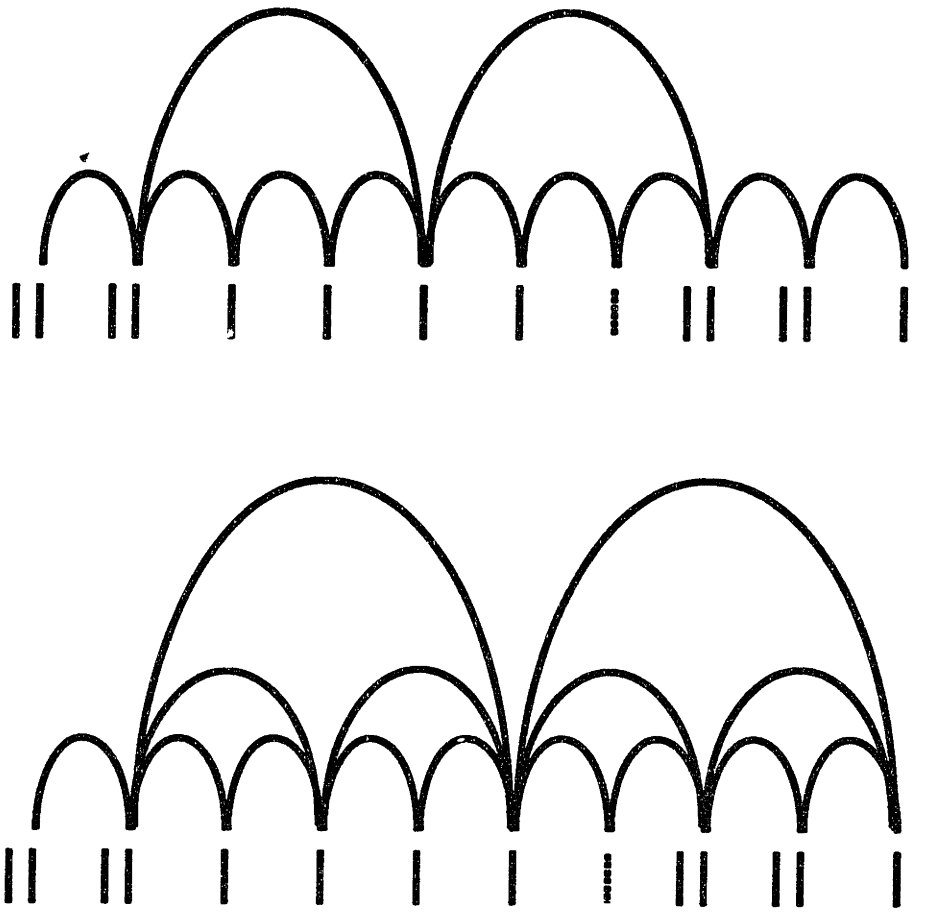
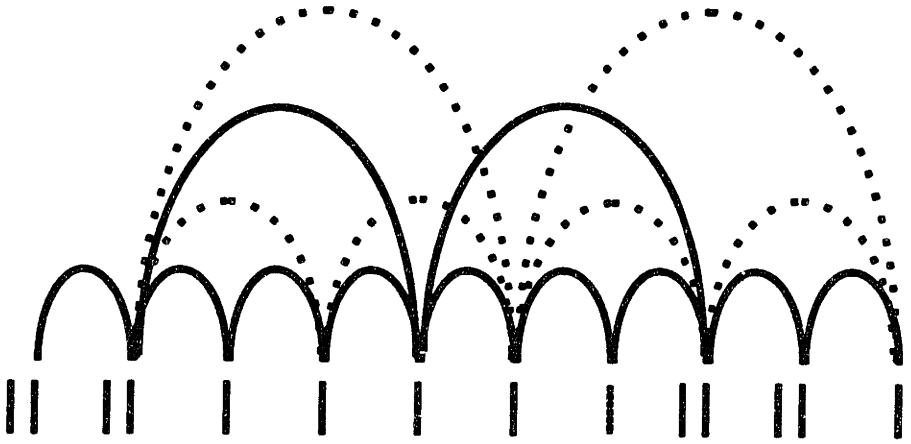
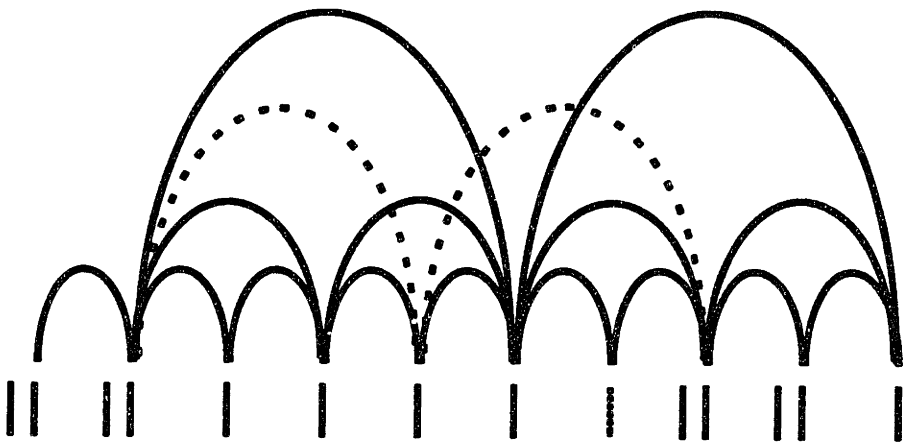


Figure Cont-10

One way would be to simply represent the two hypotheses as two separate structures of nodes connected by next, prev, parent and child links. This is not the best solution, because it is redundant; we would be maintaining two copies of the quarter-note level (the lowest level), which is identical in the two hypotheses. A better way is to have a single quarter-note level which is connected to the higher levels by context-dependent links:



context = :three-four-hypothesis



context = :four-four-hypothesis

Figure Con-11

In this scheme, the parent links of the quarter-notes are context-dependent links, and the contexts are hypotheses. In the (3 4) hypothesis, the parent links of nodes in the quarter-note level point to nodes that span three quarter notes, and in the (4 4) hypothesis, they point to nodes that span two quarter-notes (which, in turn, point to nodes that span two half-notes.)

Context-dependent links are also useful for representing split hypotheses:

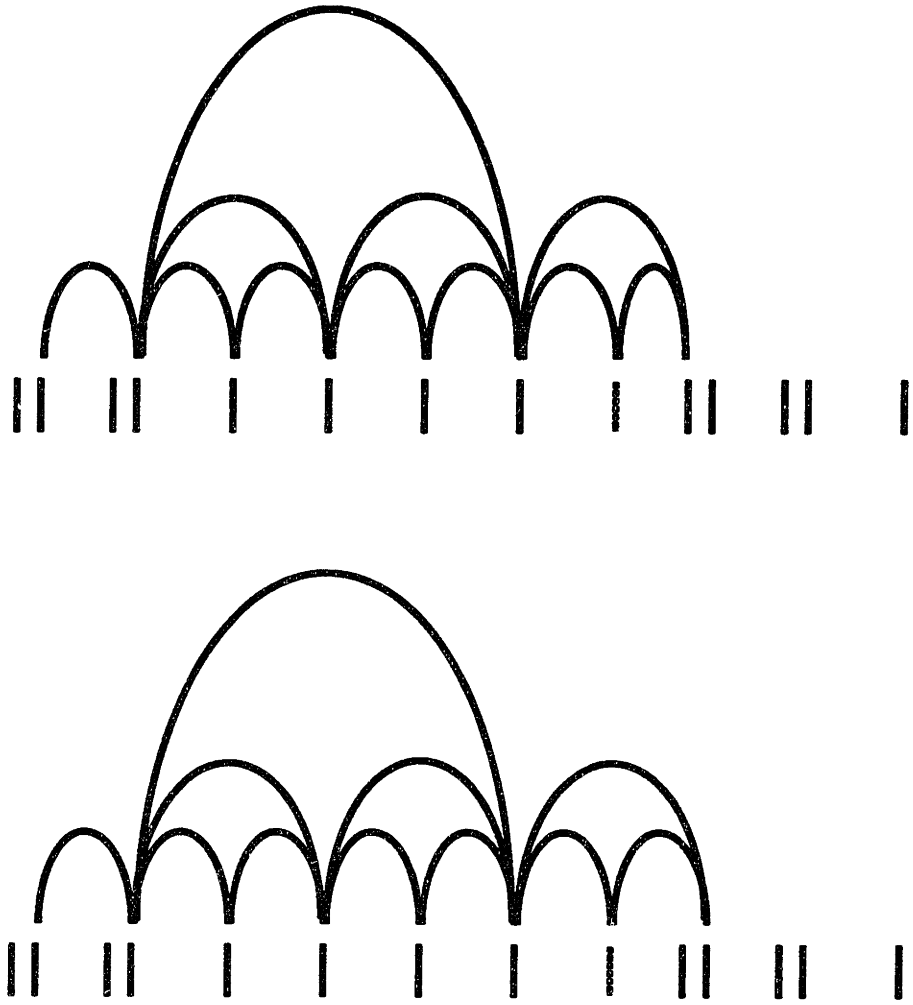
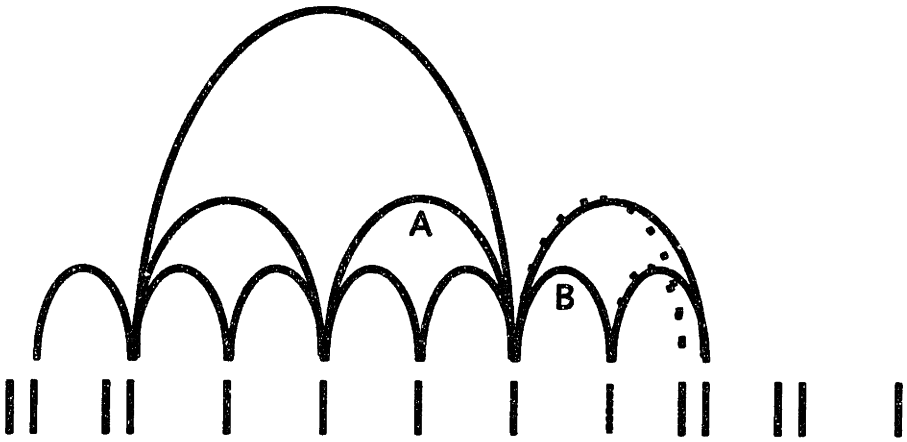
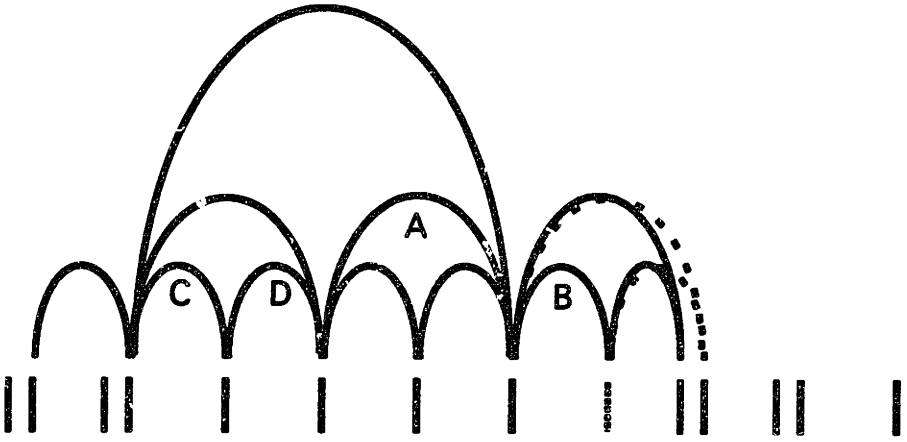


Figure Cont-12

When a hypothesis splits, most of the information in the two resulting hypotheses is the same, as shown in figure Cont-12. Again, we don't want to make two separate copies of all the nodes; we only want separate copies of the nodes that are different.



context = :hypothesis-1



context = :hypothesis-2

Figure Cont-13

This is accomplished by using context-dependent links for the "next" pointers of beat nodes. The next pointers of nodes A and B change appropriately, depending on whether *context* is :hypothesis-1 or :hypothesis-2.

Whenever we split a hypothesis, the new contexts inherit from the context of the old hypothesis. The next pointer of C will be D, in either :hypothesis-1 or :hypothesis-2. This is not because node

C knows, explicitly, where to point in either of these contexts, but because both :hypothesis-1 and :hypothesis-2 inherit from a context that node C did know about.

Part III: Related Work

Separating Sources

We have now seen a working model of one perceptual faculty — that of parsing rhythms. We now turn to the question, Can what we have learned be applied to other perceptual modeling problems? Are the various mechanisms that comprise rhythm perception also useful for other perceptual tasks, and if so, which ones, and how are they used?

It is easy to see that there are general correspondences between perceptual problems; the ideas of choosing the best of a number of interpretations, or of building increasingly refined representations are obviously applicable to a number of different kinds of situations. We will argue, however, that there are much more specific and detailed ideas that we have discussed in the context of parsing rhythm that apply to very different kinds of perceptual tasks. To that end, we will now briefly consider a different perceptual problem, that of separating sources of audio information.

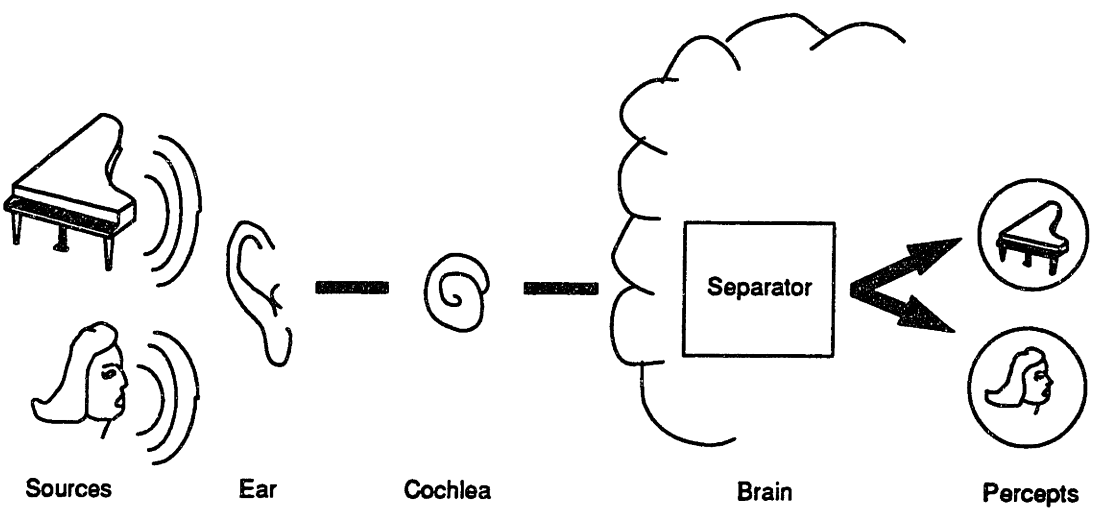


Figure Sep-1.

Figure Sep-1 is a schematic diagram of the source separation problem. The combined sound from a singer and a piano enters the ear — a single ear, since we are considering the problem of *monaural* source separation — it is then filtered by the cochlea, and eventually, separated into separate signals which correspond to the separate sources. One might be tempted to think that because humans do this so easily, that this process will be easy to emulate on a machine; as we have already seen in the case of rhythm, such a view is naive, and it is only millions of years of evolution that make the problem seem easy to us. Writing a computer program to do it is appears to be bafflingly difficult.

Ellis (Ellis 1992) has proposed and partly implemented an approach to source separation problem, which we can summarize as follows: We analyze the sound from the combined sources into a set of components which are "atomic," in the sense that they each belong to only one of the two sources. We then rebuild the separate sounds by separating the components into two groups, corresponding to the two sources.

The atomic elements of Ellis' approach are contiguous regions in a time-frequency representation of the sound. Figure Sep-2 shows such a sonogram representation of a complex sound. The x and y axes are pitch and time, respectively, and the z-axis is magnitude. Represented this way, the sound appears as a rough terrain; the atomic components, called *tracks*, are ranges of local maxima of magnitude. Tracks are depicted as squares connected by line-segments.

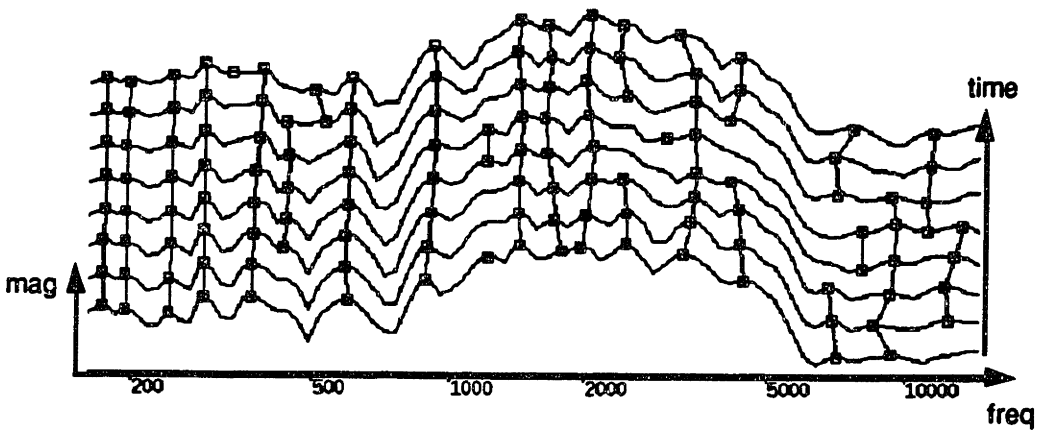


Figure Sep-2.

The key features of the tracks is that they each represent information from only one of the two sounds, and they contain enough information to reconstruct the original sounds.[1] This, combined with the assumption that each track represents information from only one source, means that source separation can be accomplished by segregating the tracks correctly.

The problem before us, then, is to automatically form these groups. As with the case of Machine Rhythm, there are a number of cues which influence how this should be done:

Common Onset: When two tracks begin at the same time, they usually originate from the same source.

[1] They also require only a fraction of the original sound's bits to represent; this fact is the basis for some efficient techniques of compressing audio data.

Harmonicity: Harmonic relations between tracks are an important cue that they belong in the same group.

Proximity: Tracks that are in the same time-frequency neighborhood are likely to be related.

Common Modulation: Tracks whose frequencies vary in the same way probably belong together.

How can we combine these cues to form groupings? Machine Rhythm provides us with a number of strategic suggestions.

1. Use low-level, computationally inexpensive heuristics to make some initial groupings, and to eliminate other groupings from consideration. The more we can do this, the fewer difficult decisions will be left later.

2. Use higher-level, more expensive routines to rank grouping hypotheses once the lower level routines have filtered out obvious losers, and left us with a (hopefully) short list of plausible candidate groupings.

3. Machine Rhythm uses a number of strategies for combining different cues into coherent hypotheses, all of which are useful in the source separation problem, where we will face the same problem at many different levels. If, in the course of working on the source separation problem, we discover still other ways of combining cues, perhaps these would in turn be useful for Machine Rhythm!

4. We will evidently have the same problems parsimoniously representing overlapping hierarchical networks that we had in Machine Rhythm. The solution used in Machine Rhythm — context-dependent links — can be used in the source separation problem as well.

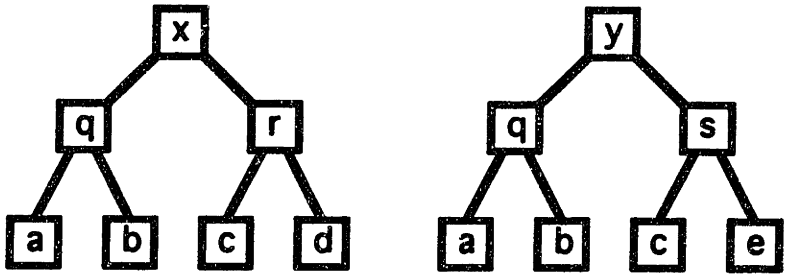


Figure Sep-3. In the hypothetical case illustrated here, we are trying to group tracks a, b, c, d, and e. We find that a and b should be grouped together, so we form the meta group q. We also find that c should be grouped with d, or c should be grouped with e, but that d should not be grouped with e. The result, in either case, r or s, should be grouped with q. This results in two different grouping hypotheses, x and y, which differ by one track. It would be inefficient to represent x and y separately; instead, we should have a means of simply switching the link to either d or e. Context-dependent links provide an appropriate mechanism for doing this.

Summary

The fact that methods used to solve the problem of parsing rhythms seem to be useful for the problem of separating audio sources should not surprise us very much. It is simply an encouraging sign that we are on the right track. It is an indication that despite the numerous ad hoc decisions that we are forced to make when designing these programs, that in the end we are still grappling with the difficult and interesting problem of understanding how perception works. The exercise of thinking about source separation helps to clarify which parts of Machine Rhythm are peculiar to the rhythm-parsing problem, and which embody more general principles.

Previous Work

This chapter discusses previous work on the rhythm parsing problem, and presents somewhat detailed descriptions of two models that I found particularly enlightening.

There have been a number of previous investigations relevant to the problem of automatically parsing rhythm, and the problem has been approached from the points of view of several different disciplines. Relevant research in psychology and linguistics includes (Bamberger 1980), (Sloboda 1983), (Lerdahl 1983), and a number of experimental results discussed in (Handel 1989). (Vercoe 1985) and (Machover 1988) discuss the application of rhythm tracking to intelligent participation by computers in live performances.

Other work has concentrated on the problem of producing computer programs or algorithms which parse rhythmic performances. Several of these investigations have viewed the problem as one of first *quantizing* the performance data and then parsing the quantized data. These investigations then centered on either the quantizing problem or the parsing problem.

Two investigations aimed at producing quantized data from live performances are those of (Schloss 1985) and (Desain 1989). The aim of Schloss' system is to transcribe percussive music. The system consists of two stages, which he terms "low level analysis" and "high level analysis." Input to the system is audio data, which the low-level analysis transforms into a MIDI-like format called a notelist. The high-level analysis then transforms the notelist into a list of multiples of a "reasonable" time unit. Schloss' system is part of a general program at Stanford which aims to transcribe music from audio data (Mont-Reynaud 1985).

Desain and Honing have applied iterative relaxation methods to the quantization problem. They have constructed a network where nodes representing the IOI's in a performance interact with each other simultaneously, altering their values so as to become simpler rational multiples of each other.

Other investigators have built systems which take quantized performances as input, and attempt to find rhythmic parsings. Two such investigations are those of Longuet-Higgins and Lee (Longuet-Higgins 1984) and Povel and Essens (Povel 1985). The basic idea behind the Longuet-Higgins and Lee model is that listeners will try to parse the rhythm so as to minimize the number of syncopations. Povel and Essens theorize that rhythmic processing is accomplished by means of an internal clock, and their model, implemented as a computer program, calculates the clock which is best induced by the input data. Although these investigators all apparently believe that

humans simultaneously perceive a number of different rhythmic levels, neither model accounts for how more than one such level would be constructed.

Investigators have also addressed the problem of producing rhythmic parsings directly from performance data; one such is that proposed in (Chung 1989). In this system, there is no intermediate step in which the timing data is quantized; the model proceeds directly from performance data to a description which is structurally close to standard musical notation, constructing a number of rhythmic levels. Chung's system processes events in strict temporal order, and is intended to model the processes employed by human listeners to find rhythms.

Another system which attempts to find rhythmic parsings directly from performances is described in (Katayose 1989).

Two Models

I have chosen two previous models of rhythm perception to describe in detail in this chapter. Each of these investigations sums up the work, over a number of years, of a group of associated researchers and benefits from their accumulated insights into the problem. The two investigations are complementary in their goals and methods; together they give a broad overview of much of the previous work in this field.

The Allan-Dannenberg Beat Tracker

The first investigation that I will summarize is described in an unpublished paper by Paul Allen and Roger Dannenberg of Carnegie-Mellon University; this work continues an investigation that also involved researchers at CCRMA, Stanford University, particularly Bernard Mont-Reynaud. The work is summarized in (Allen 1990).

The tracker works in real time, which is important for its intended applications. It processes a monophonic performance; input to the system is a stream of notes. The system assigns to each incoming note a *metric position*, which is the number of quarter-notes (or some other appropriate metric unit) that have passed by the time of the onset of that note. A particular onset, for example, might be assigned the metric position 20.5; this would mean that the onset occurred 20 quarter-notes and one eighth-note into the performance.

Dannenberg and Allan based their beat-tracker on an earlier, simpler model constructed by Dannenberg and Mont-Reynaud, to which they then refined. We will discuss the simpler model first.

The beat-tracker can be described by a *state* which is updated every time a new note is encountered. The state is represented by the tuple (T, B, S, W), where T is the time of the most recently processed note onset, B is the predicted metrical position of the most recently seen note onset, and W and S are the numerator and denominator, respectively, of δ, where $\delta = W/S$ is the *tempo*, or the number of beats per unit time. The new state is computed as follows:

$$\delta = W/S$$

$$\Delta T = T_{new} - T$$

$$\Delta B = \Delta T / \delta$$

$$Confidence = 1 - 2 \cdot \left| \Delta B - \lfloor \Delta B + 0.5 \rfloor \right|$$

$$Decay = 0.9^{\lfloor \Delta B + 0.5 \rfloor}$$

$$\Delta B = Round(\Delta B)$$

$$T' = T + \Delta T$$

$$B' = B + \Delta B$$

$$W' = W \cdot Decay + \Delta B \cdot Confidence$$

$$S' = S \cdot Decay + \Delta T \cdot Confidence$$

Let's review each of those steps in turn:

$\delta = W/S$;

$\Delta T = T_{new} - T$; these are self explanatory definitions.

$\Delta B = \Delta T \cdot \delta$; in the paper, this step is given as $\Delta B = \Delta T / \delta$, but I think that's an error. This step relates the increment in actual time to the increment in "musical time," measured in beats.

$Confidence = 1 - 2 \cdot \left| \Delta B - \lfloor \Delta B + 0.5 \rfloor \right|$; the confidence function has maxima when ΔB is an integer, and has minima when ΔB is midway between two integers — in other words, it peaks when T_{new} differs from T by an integer multiple of δ. We most expect new beats to occur at integer multiples of the established beat, and these beats are allowed to have the greatest influence in changing δ.

$Decay = 0.9^{\lfloor \Delta B + 0.5 \rfloor}$; decay is a measure of how much weight we give to old information about the tempo.

$\Delta B = Round(\Delta B)$; this step is not in the paper by Allen and Dannenberg, but I think that it's necessary. *Round* should be a function that rounds to the nearest "musical" fractional value. In other words, if the units of ΔB are quarter-notes, and the smallest beat subdivisions are eighth-notes (and there are no subdivisions between eighths and quarters), then *Round* should round to the nearest half-integer.

$T' = T + \Delta T$;

$B' = B + \Delta B$; these are self explanatory.

$W' = W \cdot Decay + \Delta B \cdot Confidence$;

$S' = S \cdot Decay + \Delta T \cdot Confidence$; we update W and S combining the information from the last beat processed and the previous beats, weighted by our confidence in each.

Suppose the onsets of the notes to be tracked are as follows: (0 100 200 305 415 530 645 760 990 1220), that is, we have a series of quarter notes whose inter-onset-interval is about 100. Between the third and the fifth quarter-notes there is a ritardando; the length of a quarter-note decreases from 100 centiseconds to 115 centiseconds. The following table describes the state after processing each new onset.

T: 100 B: 1 S: 190 W: 1.9 δ: .0100 confidence: 1 decay: .9

T: 200 B: 2 S: 271 W: 2.7100 δ: .0100 confidence: 1 decay: .9

T: 305 B: 3 S: 338.4 W: 3.3390 δ: .0099 confidence: .9 decay: .9

T: 415 B: 4 S: 395.7781 W: 3.8344 δ:.0097 confidence: .8293 decay: .9

T:530 B: 5 S: 444.9489 W: 4.2226 δ: .0095 confidence: .7717 decay: .9

T: 645 B: 6 S: 494.4386 W: 4.6176 δ: .0093 confidence: .8173 decay: .9

T: 760 B: 7 S: 542.9740 W: 5.0079 δ: .0092 confidence: .8520 decay: .9

T: 875 B: 8 S: 589.7273 W: 5.3858 δ: .0091 confidence: .8787 decay: .9

T: 990 B: 9 S: 634.1955 W: 5.7467 δ: .0091 confidence: .8995 decay: .9

T: 1105 B: 10 S: 676.1022 W: 6.0879 δ: .009 confidence: .9159 decay: .9

T: 1220 B: 11 S: 715.3253 W: 6.4081 δ: .009 confidence: .929 decay: .9

Note that the confidence dips from 1 to .7717 as the quarter-notes accelerate, and then recovers as the tempo again becomes constant. Meanwhile, the tempo goes from .01 to .009.

In processing actual performances, Mont-Reynaud and Dannenberg found that the beat-tracker did not produce satisfactory results. In their analysis, the beat-tracker was either too *sluggish*, meaning that it did not respond quickly enough to tempo changes in order to track the beat correctly, or it was *unstable*, that is, it responded too readily to apparent changes in tempo. They attempted to address this problem by parameterizing the Confidence and Decay calculations, hoping to find a setting which would work for a range of performances. This strategy still failed to produce results that they considered satisfactory.

Dannenberg and Allen then decided on a strategy of pursuing multiple hypotheses. In the system as described so far, the step $\Delta B = Round(\Delta B)$ constitutes an irrevocable decision as to the rhythmic value of the last inter-onset-interval processed. The new strategy consisted, essentially, of allowing $Round$ to produce several values which might, in a typical case, correspond to an eighth-note, a dotted-eighth-note, and a quarter-note. The system would then develop all three of the resulting states. As is the case with the Machine Rhythm system, this results in an exponential proliferation of states, and so some pruning heuristics are needed. Some of the heuristics used by Dannenberg and Allen are:

- Quarter notes should start on the downbeat or the upbeat.

- Dotted eighths should start on the downbeat or 1/4 of the way through the beat.

- Eighth-notes should begin on an even multiple of sixteenths, and if they begin 1/4 of the way through a beat, they should be preceded by a sixteenth.

These heuristics, as well as the rest of the heuristics used by Dannenberg and Allen, are generally concerned with which rhythmic values can follow others, and which are permissible in various situations. Dannenberg and Allen do not attempt to explain why these heuristics work. In general, their orientation is more toward constructing a working beat finder than building a plausible cognitive model.

Although Dannenberg and Allen were still not satisfied with the performance of their beat tracker, I found their work and that of Mont-Reynaud very helpful for clarifying my own

thinking on the subject. I had realized, as had others, that pursuing several hypotheses could be a good rhythm-finding strategy, but I hadn't managed to formulate the problem as elegantly.

The Lee Model

The model proposed in (Lee 1985) continues and refines work by Lee and a number of other researchers, particularly H. C. Longuet-Higgins. Lee's model assumes that the performance is already quantized, so that quarter-notes are exactly twice as long as eighth-notes, half as long as half-notes, and so on. The model tries, in the terminology I have developed to describe Machine Rhythm, to determine the size and location of beat-levels. The model processes a monophonic stream and relies exclusively on timing information. In contrast to Allen, Dannenberg , et al., Lee and his associates are primarily interested in building a cognitive model and in understanding musical psychology.

In a previous paper (Longuet-Higgens 1984), Longuet-Higgins and Lee proposed that listeners parse rhythms so as place long notes on stronger (that is, higher-level) beats, and that they will try to avoid syncopations. The Lee model addresses certain situations where the previous model appeared to break down. In particular, for rhythms such as that shown in figure Prev-1

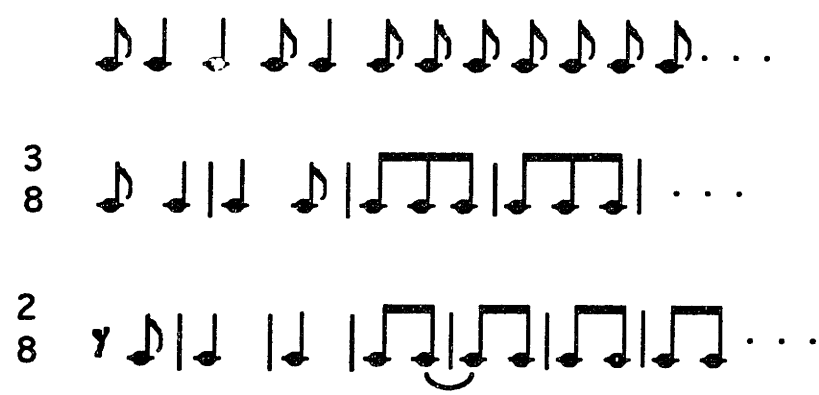


Figure Prev-1. An ambiguous rhythm with two possible parsings. The no-syncopations rule would prefer the (3 8) interpretation; Lee asserts that listeners would nevertheless parse the rhythm as (2 8) because that interpretation is established before the syncopation (in the fourth measure) occurs.

the no-syncopations rule appears not to hold.

Lee reasons that the problem with the older model is that it ignores the sequential nature of rhythm processing. Listeners do not have a global view of the rhythm that they are listening to, and can only avoid making parsings with no syncopations in so far as this is consonant with processing the rhythm sequentially in time. Once a rhythmic level has been established, it will

persist even if that leads to the perception of a syncopation. While establishing the level, however, listeners will look for an interpretation that avoids syncopations. With these considerations in mind, Lee proposes the following algorithm:

1. Set t1 to the time of the first note onset, and t2 to the time of the second note onset. Set t3 to t2 + t2 - t1, that is, so that t3 - t2 = t2 - t1.

2. If there is a notes between t2 and t3 that is longer (i.e., has a longer IOI) than the note whose onset is at t2, adjust y2 and t3 until this is no longer true — i.e., set t2 to the time of the next offset following the old t2, and again set t3 to t2 + t2 - t1.

3. We have now established a level whose beats are of length t2 - t1. Now, look for a note which is longer than this lowest-level beat, and set t1 to the time of its onset and t2 to the time of the following onset. Repeat this process, establishing higher beat-levels, for as long as there are notes which are longer then the last established beat.

We will illustrate this process with the following example.

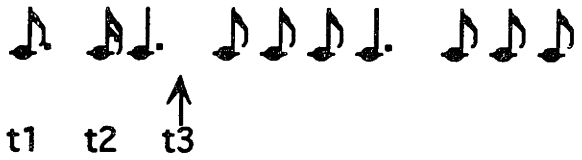


Figure Prev-2.

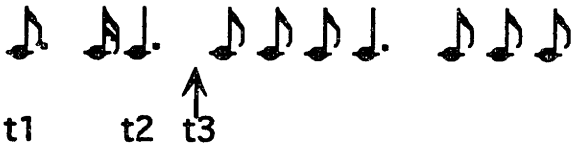


Figure Prev-3.

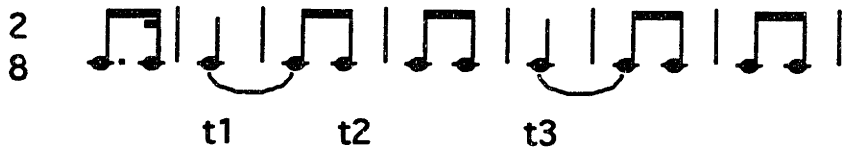


Figure Prev-4.

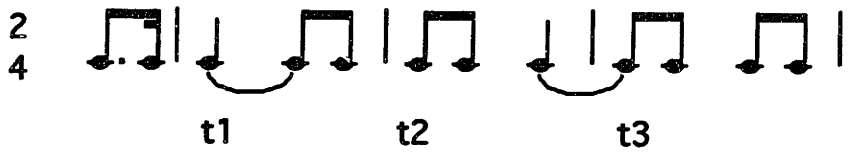


Figure Prev-5.

We begin by placing t1 on the first note, t2 on the second note, and t3 will the third and fourth notes, as in figure Prev-2. Since the third note, a dotted quarter is longer than the note at t2, a sixteenth, we adjust t2 and t3, as shown in figure Prev-3. It is no longer true that there is a note between t2 and t3 that is longer than the note on t2, so we have established our first level, the quarter-note level (figure Prev-4). To construct the next level, we place t1 on the first quarter-note beat where there is a note longer than a quarter-note, t2 on the following onset, and t3 at t2 + t2 - t1, as in figure Prev-4. Since there is a note longer than an eighth-note between t2 and t3, we must again adjust t2 and t3, as in figure Prev-5. t2 and t3 are now satisfactory, so we have established the half-note level.

Lee shows that his algorithm works on a number of examples in its intended domain — rhythmic patterns whose parsing depends on the relative lengths of notes. It is possible, however, to construct patterns where the parsing it produces seems to be at variance with the one that we produce.

Figure Prev-6.

In the example of figure Prev-6, Lee's algorithm will produce the lowest-level grouping shown in figure Prev-7.

Figure Prev-7.

It's not quite clear what humans would do in this case; I would guess that at slower tempos, human parsings would agree with Lee's algorithm, while at faster tempos, listeners would not make up their minds about the tactus level until they heard notes 5 through 7.

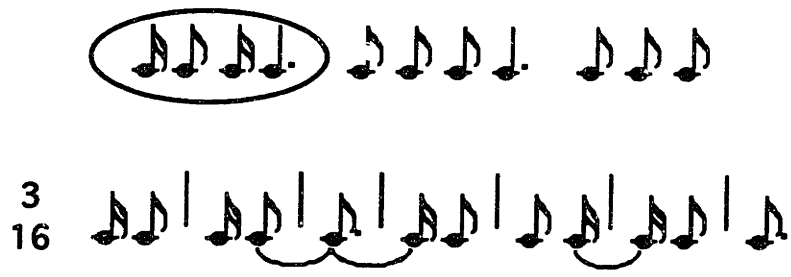


Figure Prev-8. At slower tempos, fewer notes have entered the auditory buffer (indicated by the oval) at the time that a rhythmic parsing decision has been made; in this case the listener constructs a (3 16) parsing.

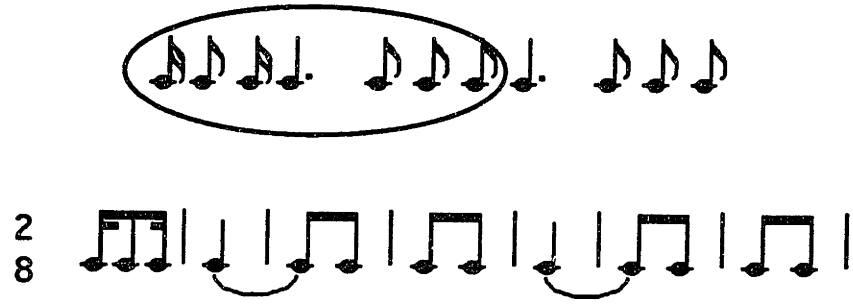


Figure Prev-9. At faster tempos, more events have entered the auditory buffer when the parsing decision is made, which in this case leads to a (2 8) parsing.

Lee's algorithm appears to suffer from lack of knowledge about the absolute tempo of the rhythm in question. His model assumes that the mind will begin trying to parse the rhythm as soon as it has two events to work with, regardless of how much absolute time that takes. In my view, Lee basically went too far in the matter of taking the sequentiality of rhythm parsing into account; Longuet-Higgins and Lee's previous model did not consider the sequential nature of rhythm parsing at all, and the Lee model discussed above considers it too strictly.

We can compromise between these two extremes, and explain the two possible parsings shown in figures Prev-8 and Prev-9, as follows. We suppose that the first few notes of the pattern are collected in a buffer whose length, in time, is T. After T units of time have passed, the mind decides on a rhythmic parsing, using the entire contents of the buffer.

Figure Prev-9 shows that at faster tempos, notes 5-7 will be included in the buffer, encouraging an eighth-note-tactus interpretation. In figure Prev-8, the tempo is slower, and notes 5-7 are not in the buffer by the time the interpretation is decided, leading to a dotted-eighth-tactus interpretation. The model that Machine Rhythm embodies is a generalization of the model proposed here.

Future Work

Not everything about Machine Rhythm is the way I would like it to be. Much of it embodies compromises with the realities of writing a large computer program. There are many problems that I didn't understand well when I wrote the code that addresses them, and have since understood better.

In this chapter I'll describe the changes and improvements that I would like to make to Machine Rhythm. Most of these improvements have the dual benefits of making Machine Rhythm a more useful and flexible rhythm parser, as well as a more convincing cognitive model. I'll refer to the proposed new version of the program as MR2.

No Preprocessor

MR2 will adhere to a stricter real-time discipline. There will not be a separate preprocessing pass, as is currently the case. The preprocessing and startup operations will be interleaved with the hypothesis extension operations, rather than placed in separate modules, as they are currently. The following example illustrates how this will work.

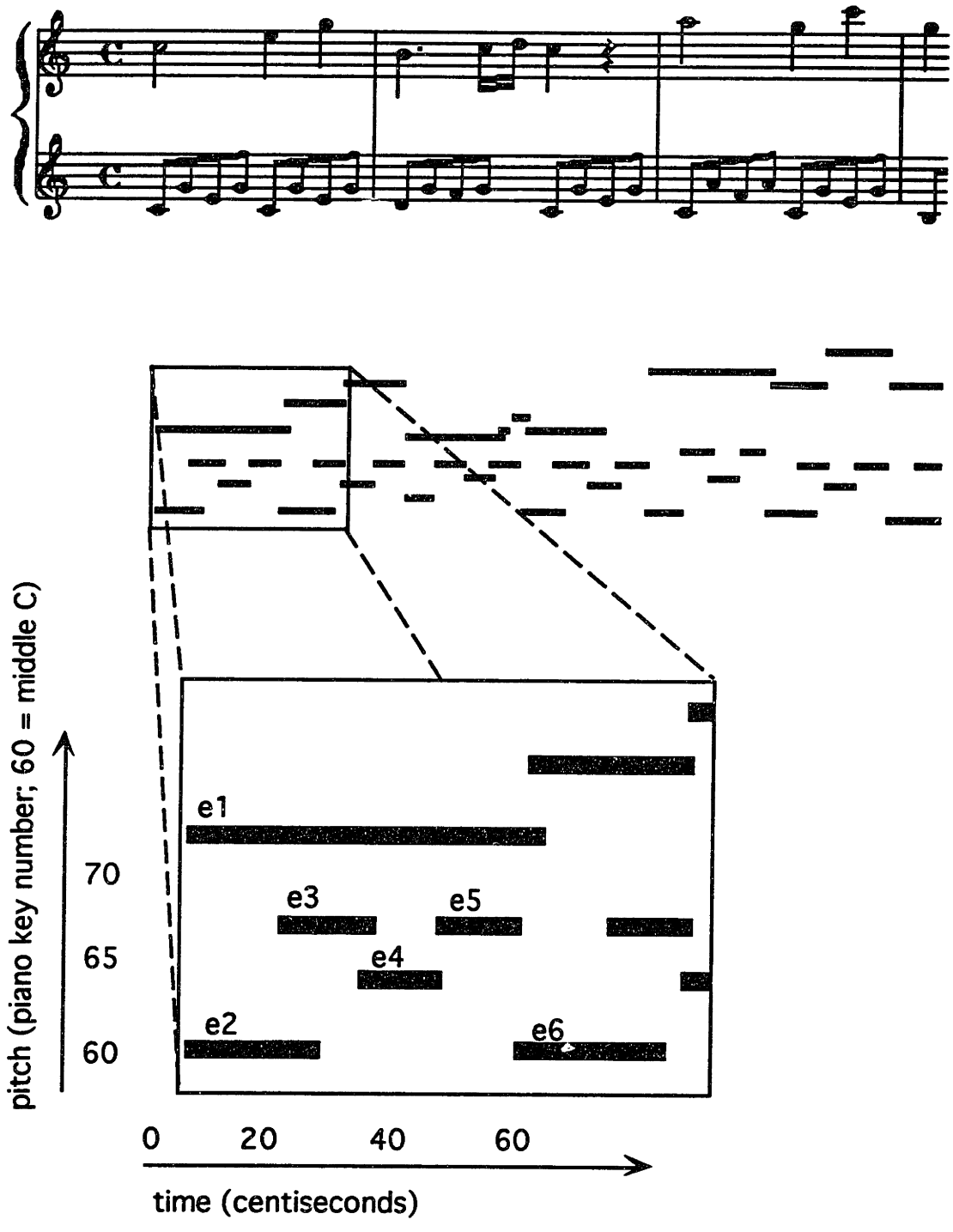


Figure Fut-1.

Input information enters the event-maker and cascades through the sequence of modules; each module passes along information to its successor. More particularly,

1. e1's onset enters the event-maker, which makes an event-object with the appropriate attributes, that is, pitch 72, time 0, velocity 56, duration and IOI indeterminate. e1 is then passed to the chord-finder, which places the event in a buffer.

2. e2's onset enters the event-maker, which creates another event, filling in the information that it can determine, and passing it along to the chord finder.

3. The chord finder notes that e1 and e2 are sufficiently close in onset time and separate in pitch so that they should be considered part of the same chord, and makes a chord object. It does not yet pass this chord to the melody finder, since there might be, for example, an event with an onset at time 2 which would also be part of the chord.

4. e3's onset enters the event-maker, which makes an event object and passes it to the chord-finder. The event-maker also sets the IOI of e2 to 20. The chord-finder now knows that there are no other events that could be part of the chord containing e1 and e2, so the construction of that chord is now finished. Furthermore, it knows that the IOI of that chord — that is, the distance from the onset of the chord to the next event — is 20 (by convention, we consider the onset of a chord to be the onset of its lowest-pitch event), so it can update the IOI of the chord appropriately. The chord is then passed to the melody-finder.

5. The melody finder has enough information to decide that e1 should be considered melody and e2 should be considered accompaniment, and passes this information to the rhythm-finder. The rhythm-finder, at this point only has one event to consider; it places that event — the chord containing e1 and e2 — in a buffer.

6. The event maker now receives the release-time for e2, at time 30, and updates e2 appropriately.

7. The event-maker receives the onset of e4, and makes the appropriate event-object. It also fills in the IOI for e3. The time of e4's onset is passed to the chord-finder, which can now determine that e3 is not part of a chord; e3 is passed to the melody finder, which classifies it as accompaniment, using the information that e1 is still sounding, since it has not yet received a release time for e1. The melody finder then passes e3 to the rhythm finder.

The rhythm finder now has two summary-events to work with; it begins to accumulate IOI's according to the histogram method discussed in the chapter "Startup." It also maintains a continuously updated best-estimate for the tactus; since there is currently only one IOI, the tactus is estimated to be 20. We also attempt to find the tactus-level on the available events; since there

are only two events, there is only one possibility for this level, which is the single arch shown in figure Fut-2

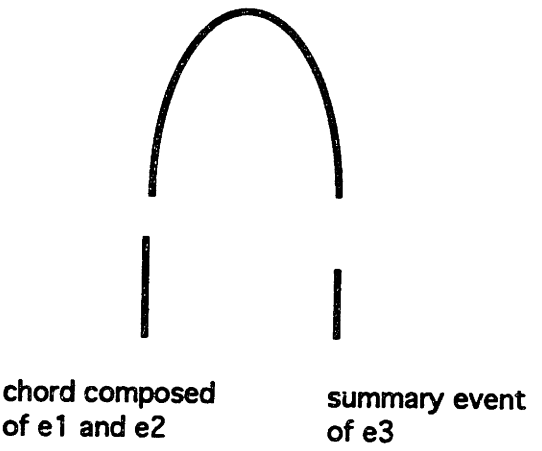


Figure Fut-2.

8. The event-maker receives the release time of e3, and makes the appropriate updates to IOI's and durations.

9. The event-maker receives e5's onset-time and constructs an appropriate event. The arrival of e5 causes e4 to be output from the chord-finder as a single event, and classified by the melody finder as an accompaniment note. e4 is passed to the rhythm-finder, which updates its estimate of the tactus, which, it turns out, is still 20. It also extends its single hypothesis, and adds a new level.

10. The arrival of e6 will cause e5 to be sent to the rhythm-finder, which will update its current hypotheses and make some new ones, as shown in figure Fut-3.

Figure Fut-3. Rhythmic hypotheses for events 1-5.

As hypotheses accumulate, some must be discarded. When that happens we rank and prune them in the same way that Machine Rhythm does.

Startup Mode

The Machine Rhythm model of rhythm parsing draws too sharp a distinction between startup and continuation. In the Machine rhythm model, the rhythm processor selects an segment at the beginning of the piece which is used to construct an initial set of hypotheses; this process happens only once. It relies on the extender to parse the rest of the piece.

In MR2, the startup and continuation processes will be integrated. The difference between them will be more a matter of degree than one of kind; when the program is in startup mode, it will devote more of its resources to generating new hypotheses, while in continuation mode, it will devote more resources to extending one or more likely candidates. Startup will, in effect, always be running; breaks in the rhythm will correspond to places where the program was unable to reconcile the hypotheses produced by the startup routine with those produced by the extender.

One of the most glaring weaknesses of Machine Rhythm is that once it has lost the correct rhythm, it is unable to recover. Integration of the startup and continuation modules is the natural way of addressing this problem.

Harmony Expert

MR2 should have more methods of choosing among hypotheses than Machine Rhythm, and those methods should be more sophisticated. One of the more obviously missing methods is one which would use harmonic information.

Changes in harmony tend to take place on higher-level downbeats, as the example in figure Fut-4 illustrates.

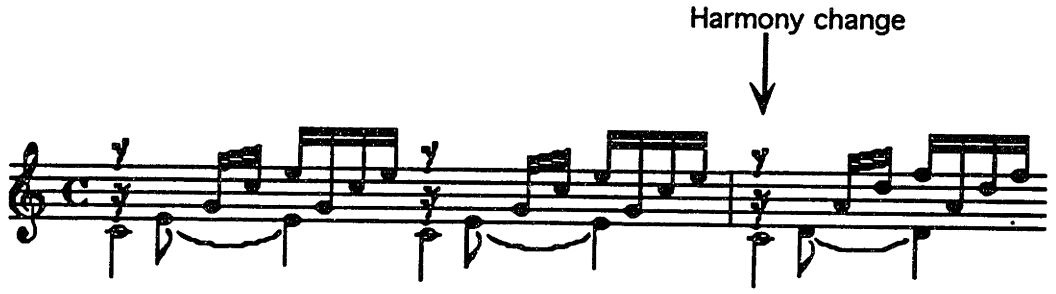


Figure Fut-4. From Prelude #1, Book I, J. S. Bach's *Well Tempered Clavier.*

MR2 will implement a specialist which will detect changes in harmony and use this information in the ranking of rhythmic hypotheses.

Better History Mechanism?

As we noted in the chapter "Previous Work," the models proposed by Mont-Reynaud and Dannenberg attached much more importance to the timing history of the performance than we have in Machine Rhythm. Their prediction of where the next beat is depends to some extent on the all of the previous beats, with the strength of their contribution depending on the value of the "history" parameter. In Machine Rhythm, the location of a new beat depends only on the interval of the current beat. Although it was neither my experience nor theirs that more complex history mechanisms made much difference in the parser's operation, it would not be difficult to make Machine Rhythm's prediction for the next beat a function of several previous beats. We could implement the Mont-Reynaud/Dannenberg scheme, or we could try others. A more sophisticated history mechanism might be particularly helpful in parsing regular changes in tempo: accelerandos and ritardandos.

Floating Bottom Level

Machine Rhythm currently makes an initial estimate of the tactus, which does not change afterwards. The tactus then remains fixed while the rest of the piece is parsed. It often happens, however, that a piece will introduce a new rhythmic subdivision after the first few measures.

MR2 will have a natural way have handling this situation; when there are too many notes that are unaccounted for by the current tactus, the program will generate an appropriate new beat level, as shown in figure Fut-5.

Figure Fut-5.

Similarly, an established lowest beat level may no longer be justified by events in the piece; in this case it should disappear, as shown in figure Fut-6.

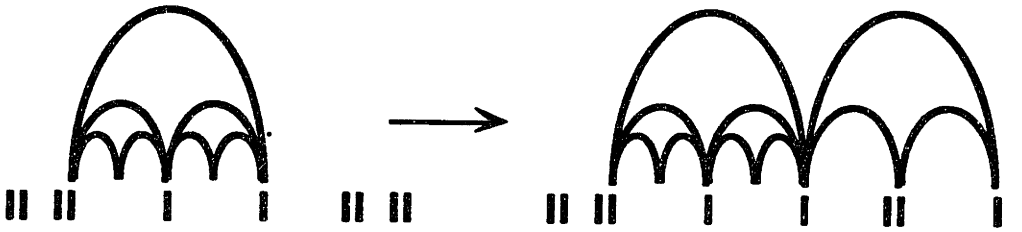


Figure Fut-6.

Dempster-Shafer Theory of Evidence

Gordon and Shortliffe have proposed a scheme for combining evidence from multiple sources, based on the Dempster-Shafer theory of evidence (Gordon 1985). It can be shown that, with appropriate assumptions, the weighted ranking scheme described in chapter "Choosing and Ranking" is equivalent to the Dempster-Shafer scheme applied to singleton hypotheses. The singleton hypothesis case, however, is only part of the Dempster-Shafer theory, and there may be other applications of the theory that MR2 could exploit; in fact, I think that in general, understanding and exploring applications of the Dempster-Shafer theory to perceptual problems could be a very fruitful future project.

A Parallel Model

In the models we have discussed up to this point, a performance is parsed by a collection of modules arranged in a fairly strict sequence. The performance, represented as MIDI bytes, is passed to the event-maker, which passes its results to the chord-finder, then to the melody-finder, then to the tactus-detector, and so on.

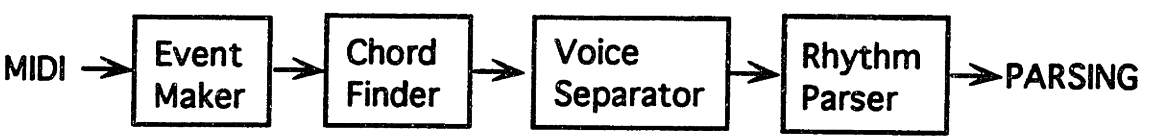


Figure Fut-7.

A program whose modules are arranged in this way has the advantage of being easy to write and debug, since the flow of information is clear, and we know how the representation should look at each stage. It may be, however, that a "messier" model better reflects what happens in human brains. Figure Fut-8 shows a different arrangement of modules.

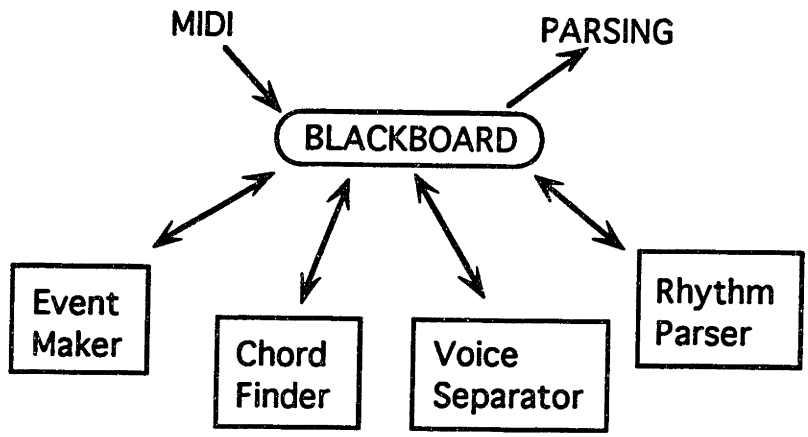


Figure Fut-8.

Here each module writes information to a central "blackboard," where it is visible to every other module. The sequence of operations on the data is not determined by the structure of the program, but rather by the structure of the problem. If in a particular case, it was difficult to find the rhythm but easy to find the melody, the system would find the melody first (perhaps after making an initial wrong pass at the rhythm) and then find the rhythm, having the benefit of a separated melody. In another case, the melody finding may be particularly difficult; in that case, the system would find the rhythm first, and then find the melody; in this case, the melody finder would benefit from the results of the rhythm-finder.

It is much more difficult to write a program in the style of figure Fut-8 than in the style of figure Fut-7. When each module is, in effect, connected to every other module, there is much greater potential for unexpected, and perhaps undesired interactions among them. It may make sense to limit these interactions — that is, to compromise between figures Fut-7 and Fut-8.

Summary

Machine Rhythm should be viewed as a step in the enterprise of understanding and modeling rhythm perception. In this chapter, we have considered further steps. MR2, Machine Rhythm's second version, should re-address real-time modeling considerations, use more sophisticated ways of deciding among alternative hypotheses, and have more flexibility to deal with changes in rhythmic structure. We should also consider other architectural arrangements of the various modules, which may more accurately reflect the architecture of human rhythm parsers.

Part IV: Evaluation and Conclusion

Evaluation

How well does the system work? In this chapter we summarize the results of a battery of tests that were run on Machine Rhythm.

I tested Machine Rhythm on a total of about 100 performances. Of these, the largest block, and the subject of the most rigorous testing, were the beginnings of 55 movements from Mozart piano sonatas, performed by Mike Hawley, another member of the Music and Cognition Group who is, among other things, a skilled pianist. I also tested the program on my own performances of most of the Mozart sonatas performed by Hawley, and, less formally, on a selection of other pieces by classical composers, and on various folk songs, national anthems, and so on. The results for the Mozart recordings are described and summarized below; the results for the other performances were roughly consistent with these.

The test of each piece consisted of two parts. First, I determined whether or not the startup module parsed the beginning of the piece correctly. I then tested how well the extension module extended the correct startup hypothesis, regardless of whether or not the startup module succeeded in producing it — so, if the startup module failed, I encoded the correct startup hypothesis by hand, and then let the extension module try to extend it. The two basic measures of Machine Rhythm's performance on the data set were:

1. The percentage of performances for which the startup module produced the correct hypothesis, and

2. The percentage of measures that it parsed correctly, given that it had parsed the previous measure correctly.

"Correct parsing," meant that the piece's rhythmic structure as determined by the program matched the parsing implicit in the score. The correct parsing did not always receive the highest rank; however, the system had to give the parsing a consistently high ranking in order to prevent it from being pruned away, since in the course of parsing a piece, the system would generally produce several hundred times as many hypotheses as it was allowed to keep. The parsing of the sixteenth-note passage from Mozart K. 545 discussed in the chapter "Choosing and Ranking" was typical. For Mike Hawley's data, the system was set to begin pruning hypotheses when their number exceeded 50, and would then prune away all but 35 of them. For the pieces I performed, those numbers were 35 and 25, respectively. I also adjusted the weight accorded to timing information so that in my performances, timing information weighed more heavily.

Results for Hawley's Performances

The overall character of Hawley's data is that they were *performances*, that is, they approximated the style that a pianist would use when giving a recital. There were very clear differences in his playing style, reflecting each movement's character, which resulted in generous expressive timing variation for some passages, and fairly metronomic playing for others. Hawley generally paused between phrases, and offset the melody and the accompaniment; both of these are interpretive techniques used by skilled pianists (Palmer 1988). As a result, this set of data thoroughly exercised Machine Rhythm's various methods for staying on the track of the correct parsing.

For Hawley's performances, the startup module found the correct starting hypothesis in 34, or 62%, or the 55 recordings. The percentage of measures parsed correctly, given that the previous measure was parsed correctly, was 95. Machine Rhythm, in other words, lost the correct parsing about once every 20 measures in this data set.

Results for Rosenthal's Performances

I also recorded my own performances of 37 of the Mozart sonata movements. My performances were on the whole somewhat more evenly timed than Hawley's, and somewhat easier for Machine Rhythm to parse. I believe this is due to several factors. First, I obviously wanted the program to work; it would be disingenuous to try to claim that this had no effect on the way I played. Also, my performance style is different from Hawley's (his is a bit on the romantic side of my own), and I'm not as skilled a pianist. I obtained Hawley's performances to at least partly obviate concerns about objectivity.

For my performances, the startup module found the correct parsing in 24, or 65% of the 37 performances. 98.5% of the measures were parsed correctly, given that the previous ones were parsed correctly, in other words, the program lost the correct parsing once every 67 measures.

Short descriptions of each test are included in Appendix B.

Conclusion

> What magical trick makes us intelligent? *The trick is that there is no trick.* The power of intelligence stems from our vast diversity, not from any single perfect principle. Our species evolved many effective although imperfect methods, and each of us as individuals develops more on our own. Eventually, very few of our actions comes to depend on any single mechanism. (Minsky 1986, p. 308)

A curious thing about this project is that Machine Rhythm started successfully parsing rhythms rather soon after it was first put together. At first I gave myself the credit, and decided that my programming skills must be improving. But it gradually became clear that even though the program appeared to be working, rather large sections of it weren't doing what they were supposed to do at all. While I was sorry to have to revise my own estimation of my programming ability back downwards, I was consoled that this seemed to indicate something fundamentally right about the design of the program. In short, it was not relying too much on any particular part of itself, and that was what I was trying to accomplish in its overall design.

A little later in *The Society of Mind*, Minsky writes

> To assemble the overview suggested in this book, I had to make literally hundreds of assumptions (Minsky 1986, p. 322).

and so it is with this thesis — hundreds sounds about right — and, of course, this thesis concerns at best only a tiny fraction of the mind. Fortunately, however, there's only one really crucial assumption, and that's the fundamental one that the mind doesn't rely very heavily on any of its parts. That means that a theory of mind that starts out with that view won't rely too heavily on any of *its* possibly faulty parts. In physics, one wrong assumption can be fatal, but good theories of mind should be more robust, like the minds they describe.

In the end, I think that this idea was the guardian angel of this project. The project certainly needed a guardian angel, since no one had any idea of how difficult a project it would prove to be, or how long it would take to produce any results worth writing about. The worth and interest of this project are not that I got it right, but rather that I probably got quite a bit of it wrong, and it works anyway.

Appendices

A: The Whirlwind Tour

This chapter takes the reader through a brief summary of this thesis, a "whirlwind tour." We will introduce the basic concepts, often relying heavily on the reader's ability to fill in logical gaps and intuit tacit understandings; we will then summarize the main stages of the program's workings. For the casual reader, this chapter can be viewed as a way to get a quick overview of the thesis. For the more serious reader, this chapter should raise a number of questions; answers to those questions can hopefully be found in the main body of the thesis.

Basic Concepts

When people listen to a rhythmic piece of music, such as a march or a dance, they can indicate their perception of the piece's rhythm by tapping their hand or foot. So, for example, when listening to "La Marseillaise," one taps one's finger as indicated by the arches in the diagram below.

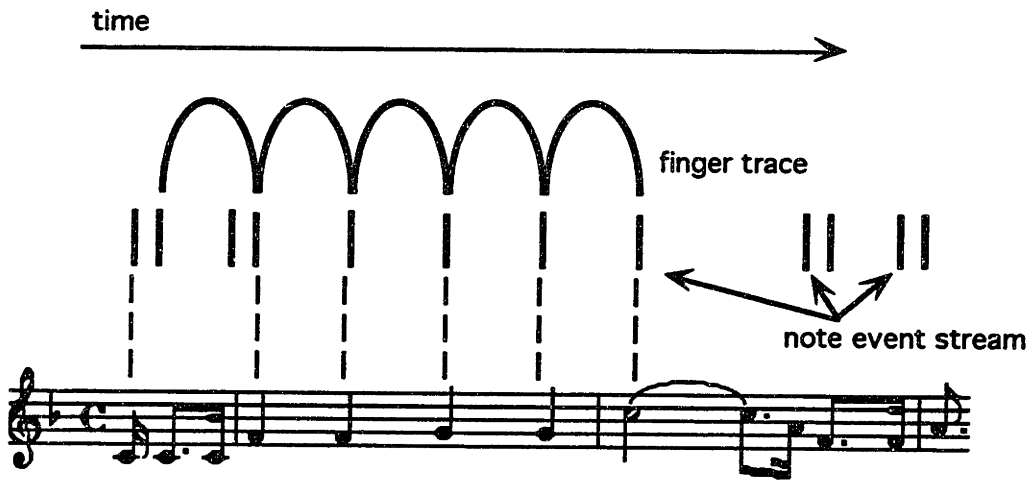


Figure A-1 Tapping the quarter notes of "La Marseillaise." The arches represent a trace of the listener's finger, and the vertical bars represent note events. Each note event by a vertical bar positioned at the point in time corresponding to the note's onset.

Musicians would refer to this as "tapping the quarter note level." One could also tap the "half-note" level, that is, tap every other quarter note, as indicated by the larger arches below:

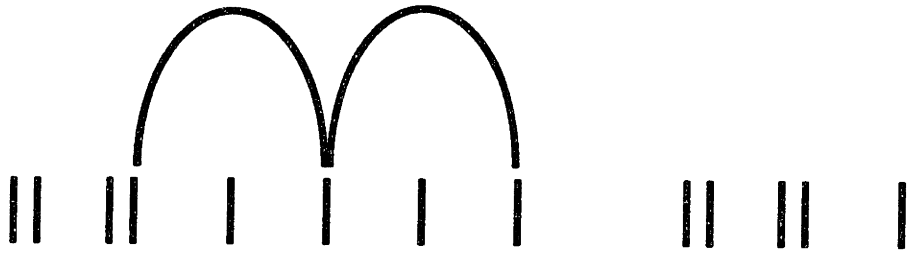


Figure A-2. Tapping the half-notes

and one could even, in principle, tap the whole-note, or measure level:

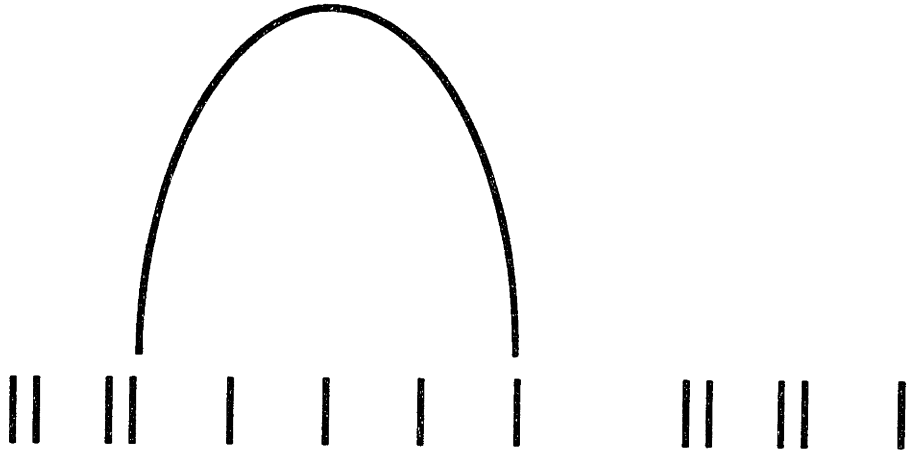


Figure A-3. Tapping the measure level.

In other words, the rhythmic structure that listeners build in their minds consists of a hierarchy of *levels*, which we can represent with the following kinds of pictures.

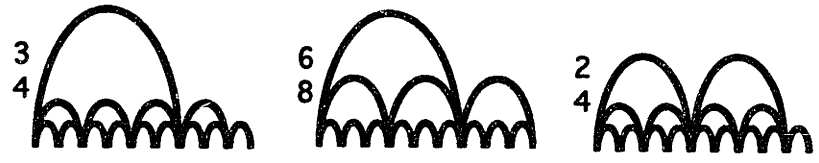


Figure A-4. Rhythmic hypotheses. Each rhythmic hypothesis corresponds to a time signature.

We will call these structures *hypotheses*. The individual arches that compose levels are called *beats;* the width of each arch, corresponding to the length in time of the beat, is called the beat's *interval*. Certain formal constraints apply to hypotheses; for example, in a single structure, the notes pointed to by a larger level are a subset of those pointed to by a smaller level. Another constraint is that the beats at each level must be temporally sequential, i.e. the arches cannot cross. The figure below shows this constraint being violated.

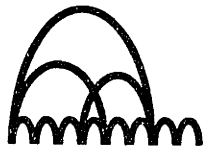


Figure A-5. Ungrammatical hypothesis.

It is usually the case that adjacent arches are subdivided by the same number of smaller arches; when this property holds, we call the hypothesis *isomeric*. The most common non-isomerism in music is a change from subdivision by two to subdivision by three; the musical term for this situation is *triplet*.

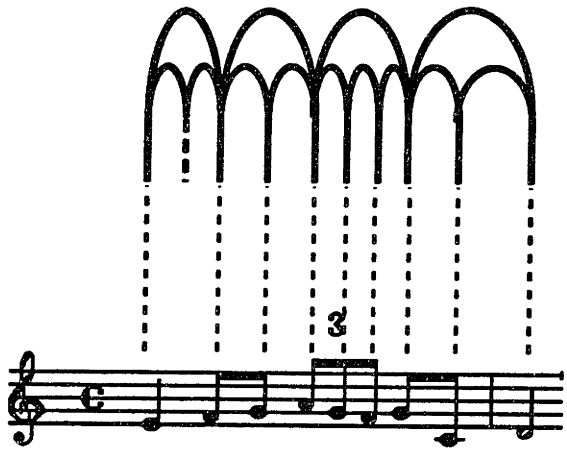


Figure A-6.

The next few sections provide a overview of the main stages through which the program proceeds as it parses the rhythm of a performance.

Preprocessing

Input to the program is a sequence of MIDI bytes representing a musical performance (most typically, but not necessarily, of a piano piece). Before the program can begin to try to find the rhythm there are some preprocessing operations which reorganize the MIDI data into a form more suitable for rhythm finding. First, the MIDI bytes are organized into *events*, which correspond to musical notes. Second, the notes are organized into *summary events*, these being events which are considered *effectively simultaneous*. In performances of polyphonic music, some sets of notes are perceived as having simultaneous onsets; the most common example of this is a chord. In fact, the notes of a chord are not really simultaneous; they are usually spread over a period of time of up to about forty milliseconds. Whether or not a set of notes is

perceived as simultaneous depends on a number of factors, such as how far apart their pitches are, how long they are sustained, and how close together their onsets are.

Having grouped the notes into summary events, the program attempts to separate the *top voice*. Here too, the program must emulate a perceptual facility of humans, in this case, their ability to organize groups of sounds into *streams*. In music, these streams are called *voices*. It is often advantageous in rhythm finding to be able to classify notes according to whether they belong to the melody or the accompaniment. Further on in the processing the program makes use of this classification.

The Main Modules

We are now ready to begin the rhythm-finding proper. Three main modules comprise the program: *startup*, where we form a set of rhythmic hypotheses at the beginning of the piece, *extension*, where the remainder of the piece is interpreted according to a given hypothesis (which may *split*, forming new hypotheses), and *ranking*, where hypotheses are ranked according to the likelihood that they represent the interpretation humans would make. Ranking serves two purposes; first, the top-ranked hypothesis represents the program's best guess as to what the rhythm of the piece is; second, it allows the program to prune away the less likely hypotheses. This is important because the number of hypotheses increases exponentially with the number of notes processed. We will look at each of these processes in turn.

Startup

The aim at this stage is to construct every plausible hypothesis for the first few seconds of the piece, and then to rank the hypotheses according to likelihood that they are "correct," i.e. preferred by human listeners. The first step in this process is to find the *tactus*, that is, the smallest beat-interval which will concern us; this typically corresponds to an eighth-note or a quarter-note. This is accomplished by a process that forms histograms of note inter-onset-intervals; the actual algorithm employed resembles that developed by Minsky, Gold and Rabiner for pitch-tracking (Gold 1969).

Having found the tactus, we sequentially search for a set of notes which comprises the tactus level. We then build every plausible hypothesis with the tactus level as its lowest level:

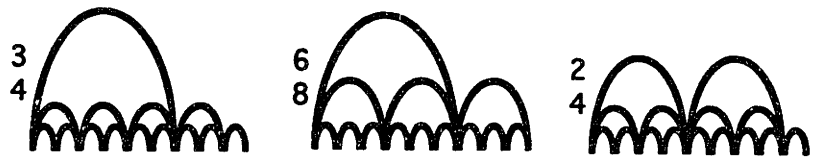


Figure A-7. Alternative hypotheses built on the same tactus.

At this stage, if all has gone well, we have a collection of hypotheses about the rhythm of the first few seconds of the piece, one of which represents "the" rhythm of the piece, that is, the interpretation that a human listener would construct.

Hypothesis Extension

This module extends a hypothesis to include yet-unprocessed notes of a performance:

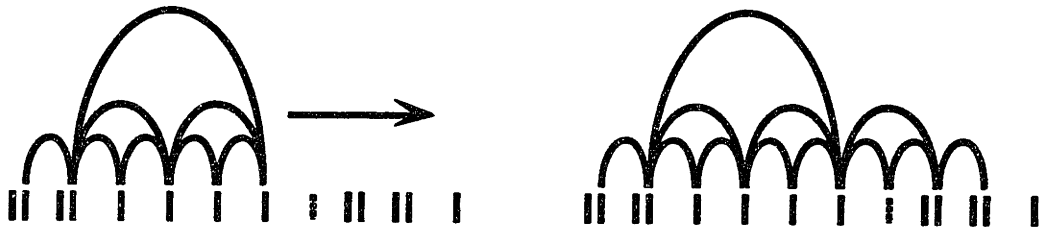


Figure A-8.

This is accomplished as follows: Each beat examines a time-window surrounding the point in time at which it expects the next beat:

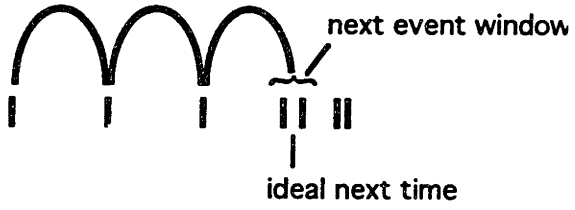


Figure A-9.

Inside this window there is some number (possibly zero) of new notes. If there are no new events, we construct a *ghost-event*. This represents a point in the performance where one would tap one's finger but no note sounds. In the general case that there is more than one event, we have to choose the one which best continues the beat. This is the task of a module called *choose*, which is described in briefly the next section (and in great detail in the chapter "Choosing and Ranking"). It may happen that choose cannot determine a single event which best continues the level, in which case the hypothesis containing the level splits to produce two new hypotheses.

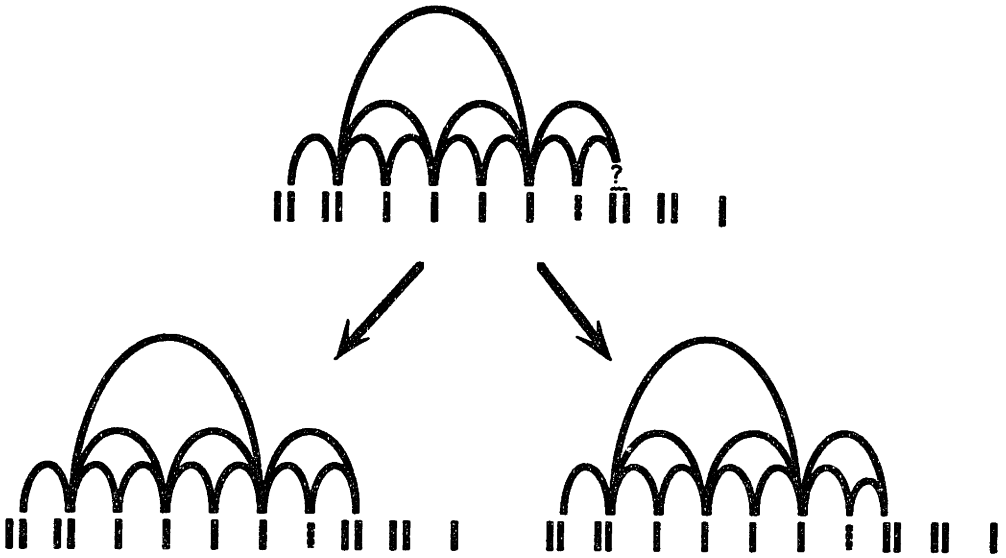


Figure 10.

In general, this process is taking place simultaneously at all of the levels that compose a hypothesis; most of the work of the extension module is bookkeeping required to to keep these processes coordinated.

Choosing and Ranking

Choose and *rank* are modules whose responsibility it is to decide which of a number of objects is preferred. In the case of *choose*, the objects are individual notes; for *rank* the objects are hypotheses. The design of these modules is adapted from the discussion of perception in *The Society of Mind* (Minsky, 1986). The *choose* and *rank* modules are discussed together here, because they both use a similar strategies to make their decisions. Those strategies might best be termed *management of multiple sources of evidence*, and may be summarized as follows:

Human rhythm perception depends on a number of different kinds of cues. In some passages we are mainly cued by the *timing* of the notes; the relations between longer and shorter notes tell us what the rhythm is. In other passages the notes may all be of the same length, so that the timing tells us nothing, but a repeating pattern in the melody indicates the rhythm. The density of chords may provide a cue, as may the locations of changes in the harmony. For any one of these rhythm finding methods, there are musical situations in which it will not work — that is, it will fail to indicate the rhythmic hypothesis that a human listener would in fact choose.

To model this situation, we construct a number of programmatic *specialists*, each of which specializes in one of these rhythm-choosing methods. There is for example a *timing* specialist, which prefers hypotheses in which adjacent beats are very close to the same length. The timing specialist is usually right, but it can also fail, especially when there is an expressive deviation in the performance from metronomic time. Other specialists look for repeating motivic patterns in the melody or accompaniment. Still others might be termed "anti-syncopation" specialists, since they tend to prefer interpretations in which there are fewer syncopations. In the current version of the program, six such specialists are used in the *rank* module and three in the *choose* module.

Each of these specialists ranks the objects under consideration according to its criterion. It is then necessary to integrate their results. In the rare situation that the specialists all agree on which interpretation is best, the choice is obvious. In general, the specialists disagree, and we use various strategies for deciding which are correct:

1. Some specialists are good at making some distinctions but not others. The timing specialists , for example, is good at telling us which of several (4 4) interpretations is correct, but useless for choosing between (3 4) and (4 4).

2. Some specialists work well in conjunction with other specialists . When the timing and anti-syncopation specialists agree, we can be fairly sure that they are both right.

3. Specialists differ not only in the overall reliability but also in their ability to avoid false positives. The melodic-pattern specialist often produces no result at all, but when it does have a preference, it is relatively reliable.

4. When we can't do anything else, we simply vote. Each specialist has been assigned a weight — essentially, the number of votes it can cast for its preference.

The specialists themselves employ teams of subspecialists to produce their results. The anti-syncopation specialists employs two different ways of detecting syncopations, and the timing specialist computes a weighted sum of timing variations at several different levels. The overall design of the program includes an established protocol for adding new specialists and improving the reliability of existing ones. There is currently no harmonic-change specialist for example; I hope to add one in the future.

Applications

The program can produce, at any point in the performance, the currently-best-ranked hypothesis, by calling the rank module. This result is interesting in itself, since it can be compared with a human listener's parsing of the performance. As with the human's parsing, the program's result can also be applied in various ways. The most obvious application is transcription of the performance into standard musical notation; so, for example, the program can be used to input a performance to a score editing system.

Another application is the synchronization of independently performed audio tracks. Suppose, for example, that a performer separately records the *primo* and *secundo* parts of a piano piece for four-hands. If we simply start both recordings at the same time, the two parts will not be synchronized, and the result is cacophonous, even if the performance is relatively metronomic. The situation does not improve much if we globally "stretch" one of the performances so that they both last the same amount of time; the performers natural deviations from metronomic time will still result in unacceptable asynchrony. But if we use the rhythmic parsing produced by our program, we can synchronize the two parts intelligently, stretching or compressing each beat as necessary. The resulting performance sounds as though the two parts had been played together.

B: Test Log

All performances are of Mozart piano sonatas. "Continues without difficulty" means that the program extended a correct hypothesis for the first measure (given to it either by the startup module or the user) across the entire example. "Lost the correct hypothesis in measure N" (or a variant of that) means that the correct hypothesis was pruned away in measure N.

Rosenthal's Performances

K. 330 1st movement, bars 1-8
Startup: A hypothesis with 16th and 8th-note levels ranks 1st. Its quarter-note level has incorrect phase, misled by syncopation.
Continues without difficulty, despite substantial timing variation.

K. 330 2nd movement, bars 1-8
Startup: A hypothesis with correct 8th and quarter-note levels ranks 1st; does not make measure level.
Continues without difficulty.

K. 330 3rd movement, bars 1-17
Startup makes correct hypothesis, ranks it 1st.
Continues without difficulty.

K. 547a 1st movement, bars 1-16
Startup fails.
Correct hypothesis is lost in measure 3, but continues correctly if alternative subdivision is turned off.

K. 279 1st movement, bars 1-10
Startup: 8 hypotheses tie for 1st, including the right one.
Continues without difficulty.

K. 279 2nd movement, bars 1-6
There aren't enough startup events to make the correct (9 8) hypothesis; a (6 8) hypothesis ranks 2nd.
Continues without difficulty, only making one hypothesis.

K. 279 3rd movement, bars 1-10

Startup finds correct hypothesis, ranks it 3rd.

Continues without difficulty.

K. 280 1st movement, bars 1-16

Startup produces 5 hypotheses tied for 1st, including a correct one, with quarter notes as its highest level.

Continues through measure 10 correctly, except for a spurious triplet in measure 2, but loses the correct hypothesis in measure 11. Stays on track impressively in measures 8-10.

K. 280 2nd movement, bars 1-8

Startup fails; the histogram decides on a 16th-note tactus, level-constructor can't construct one.

Continues without difficulty.

K. 280 3rd movement, bars 1-16

Startup finds hypothesis which is correct but contains an extra level. It's tied for 1st.

Continues without difficulty.

K. 332 1st movement, bars 1-12

Startup finds correct hypothesis and ranks it 1st.

Continues without difficulty.

K. 332 2nd movement, bars 1-4

Startup finds correct hypothesis with quarter-notes as highest level, ranks it 1st.

Continues without difficulty.

K. 332 3rd movement, bars 1-14

Startup fails.

Continuation loses correct hypothesis in measure 9.

K. 281 1st movement, 1-8

Startup finds correct hypothesis (which, however, starts in middle of measure).

Continues without difficulty, despite numerous alternations between triplets and duplets.

K. 281 2nd movement, bars 1-14

Startup finds correct hypothesis and ranks it 1st.

Continues without difficulty, including correct parsing of triplets.

K. 281 3rd movement, bars 1-12

Startup found the correct hypothesis and ranks it 2nd; did not make an eighth-note level.

Continues without difficulty.

K. 331 1st movement, bars 1-8

Startup failed.

Continues without difficulty.

K. 331 2nd movement, bars 1-10

Startup failed.

Continues without difficulty.

K. 331 3rd movement, bars 1-8

Startup finds correct hypothesis and ranks it 1st.

Continues without difficulty.

K. 333 1st movement, bars 1-10

Startup produces correct hypothesis, which is in a 4-way tie for 1st.

Continues without difficulty.

K. 333 2nd movement, bars 1-8

Startup produces correct hypothesis with quarter-notes as highest level, and ranks it 2nd.

Continues without difficulty.

K. 333 3rd movement, bars 1-16

Startup produces correct hypothesis with quarter-notes as highest level.

Continues without difficulty, including correct triplet parsing.

K. 309 1st movement, bars 1-8

Startup: starts up a (correct) (2 4) hypothesis beginning in measure 3.

Continues without difficulty, despite substantial timing variation.

K. 309 2nd movement, bars 1-5

Startup fails.

Continues without difficulty.

K. 309 3rd movement, bars 1-8
Startup finds correct hypothesis and ranks it 1st.
Continues without difficulty.

K. 570 1st movement, bars 1-12
Startup finds correct hypothesis with quarter notes as highest level, and ranks it 1st.
Continues without difficulty.

K. 570 2nd movement, bars 1-4
Startup finds correct hypothesis with quarter notes as highest level, and ranks it 1st.
Continues without difficulty.

K. 570 3rd movement, bars 1-8
Startup finds correct hypothesis and ranks it 1st.
Continues without difficulty.

K. 331 1st movement, bars 1-7
Startup fails.
Continues without difficulty.

K. 331 2nd movement, bars 1-8
Startup finds correct hypothesis with quarter notes as highest level, and ranks it 2nd.
Continues without difficulty.

K. 331 3rd movement, bars 1-8
Startup finds correct hypothesis and ranks it 1st.
Continues without difficulty.

K. 576 1st movement, bars 1-8
Startup finds correct hypothesis and ranks it 2nd.
Continues without difficulty.

K. 576 2nd movement, bars 1-8
Startup fails.

Continuation loses correct hypothesis in measure 7.

K. 567 3rd movement, bars 1-16
Startup finds correct hypothesis and ranks it 1st.
Continues without difficulty, including triplets.

K. 284 1st movement, bars 1-9
Startup fails.
Continuation loses correct hypothesis in measure 8 when given 16th notes as lowest level, but continues without difficulty when lowest level is 8ths.

K. 284 2nd movement, bars 1-12
Startup fails.
Continuation loses correct hypothesis in measure 12.

K. 284 3rd movement, bars 1-8
Startup finds correct hypothesis and ranks it 1st.
Continues without difficulty.

Hawley's Performances

K. 279 1st movement, bars 1-10
Startup fails.
Continuation loses correct hypothesis in 3rd measure.

K. 279 2nd movement, bars 1-6
Startup finds correct hypothesis which does not include measure level, ranks it 1st.
Continues without difficulty.

K. 279 3rd movement, bars 1-10
Startup produces correct hypothesis, in 3-way tie for 1st.
Continues without difficulty.

K. 280 1st movement, bars 1-12.
Startup produces correct hypothesis with quarter-notes as highest level, ranks it 1st.
Continuation loses correct hypothesis in measure 11.

K. 280 2 movement, bars 1-6.
Startup produces correct hypothesis, ranked 1st, which starts in second measure.
Continuation can't track it, however.

K. 280 3rd movement, bars 1-6
Startup produces correct hypothesis. which ties for 1st.
Continues without difficulty.

K. 281 1st movement bars 1-8
Both startup and continuation fail.

K. 281 2 movement, bars 1-14
Startup produces correct hypothesis, which ranks tied for 1st.
Continues without difficulty, despite substantial rubato.

K. 281 3rd movement, bars 1-12
Startup produces hypothesis whose phase is off at the measure level; when listening, I make the same mistake.
Continues without difficulty.

K. 282 1st movement, bars 1-8
Startup fails.
Continuation loses correct hypothesis in measure 4.

K. 282 2 movement, bars 1-12
Startup fails.
Continues without difficulty.

K. 282 3rd movement, bars 1-7
Startup produces correct hypothesis, with 8th-notes as highest level, which ties for 1st.
Continuation loses correct hypothesis in measure 6.

K. 282 4th movement, bars 1-8
Startup produces correct hypothesis, which ties for 2nd.
Continues without difficulty.

K. 283 1st movement, bars 1-10
Startup produces correct hypothesis, which ranks tied for 1st.
Continues without difficulty.

K. 283 2nd movement, bars 1-4
Startup produces correct hypothesis, with quarter-notes as top level, which ranks 2nd.
Continues without difficulty.

K. 283 3rd movement, bars 1-24
Startup produces correct hypothesis, with 8th notes as highest level; it ties for 1st.
Continuation loses the correct hypothesis in measure 9.

K. 284 1st movement, bars 1-9
Startup fails.
Continues without difficulty.

K. 284 2nd movement, bars 1-16
Startup fails.
Continuation is confused by triplets in measure 9.

K. 284 3rd movement, bars 1-8
Startup produces correct hypothesis whose highest level is quarter-notes, which ties for 1st.
Continues without difficulty.

K. 309 1st movement, bars 1-14
Startup fails.
Continuation loses the correct hypothesis in measure 11.

K. 309 2nd movement, bars 1-8
Startup fails.
Continues without difficulty.

K. 309 3rd movement, bars 1-8
Startup produces correct hypothesis, which ranks tied for 1st.
Continues without difficulty.

K. 310 1st movement, bars 1-8
Startup produces correct hypothesis, which ranks tied for 1st.
Continues without difficulty.

K. 310 2nd movement, bars 1-8
Startup produces correct hypothesis, with quarter-notes as top level, which ranks 3rd.
Continues without difficulty.

K. 310 3rd movement, bars 1-20
Startup produces correct hypothesis, which contains an extra ternary level, and ties for 1st.
Continues without difficulty.

K. 311 1st movement, bars 1-7
Startup fails.
Continues without difficulty.

K. 311 2nd movement, bars 1-8
Startup produces a correct hypothesis, with no measure level, which is ranks 2nd.
Continues without difficulty, despite long pause.

K. 311 3rd movement, bars 1-8
Startup produces correct hypothesis and ranks it 3rd.
Continues without difficulty.

K. 330 1st movement, bars 1-12
Startup fails.
Continues without difficulty.

K. 330 2nd movement, bars 1-8
Startup fails.
Continuation loses correct hypothesis in measure 12.

K. 330 3rd movement, bars 1-20
Startup produces correct hypothesis, which is in 4-way tie for second in startup.
Continuation loses correct hypothesis in measure 6.

K. 331 1st movement, bars 1-8
Startup fails.
Continuation fails.

K. 331 2nd movement, bars 1-8
Startup produces correct hypothesis, which ranks 2nd.
Continues without difficulty.

K. 332 1st movement, bars 1-12
Startup fails.
Continues without difficulty.

K. 332 2nd movement, bars 1-4
Startup produces correct hypothesis, which ranks 1st.
Continues without difficulty.

K. 332 3rd movement, bars 1-13
Startup fails.
Continuation fails.

K. 333 1st movement, bars 1-10
Startup fails.
Continuation loses correct hypothesis in measure 6, despite brave effort.

K. 333 2nd movement, bars 1-8
Startup produces correct hypothesis, which ranks 1st (only one hypothesis).
Continuation loses correct hypothesis in measure 7.

K. 333 3rd movement, bars 1-16
Startup produces correct hypothesis, which ranks 3rd.
Continues without difficulty.

K. 457 1st movement, bars 1-8
Startup produces correct hypothesis, which ranks 2nd.
Continues without difficulty.

K. 457 2nd movement, bars 1-4
Startup produces correct hypothesis, which ranks 1st; highest level was quarter note.
Continuation loses correct hypothesis in measure 4.

K. 457 3rd movement, bars 1-16
Startup fails.
Continues without difficulty.

K. 494 1st movement, bars 1-12
Startup produces correct hypothesis, which ranks 2nd
Continues without difficulty.

K. 533 1st movement, bars 1-8
Startup fails.
Continues without difficulty.

K. 533 2nd movement, bars 1-10
Startup fails.
Continues without difficulty.

K. 545 1st movement, bars 1-12
Startup produces correct hypothesis, which ranks 1st.
Continuation ran out of memory in measure 7.

K. 545 2nd movement, bars 1-8
Startup produces correct hypothesis, which ranks 1st; it did not include measure level.
Continuation loses correct hypothesis in measure 7.

K. 545 3rd movement, bars 1-8
Startup produces correct hypothesis, which ranks 2nd.
Continues without difficulty.

K. 547a 1st movement, bars 1-16
Startup fails.
Continues without difficulty.

K. 547a 3rd movement, bars 1-8
Startup produces correct hypothesis, which ranks tied for 4th.
Continues without difficulty.

K. 570 1st movement, bars 1-12
Startup produces correct hypothesis, which ranks tied for 1st.
Continues without difficulty.

K. 570 2nd movement, bars 1-4
Startup fails.
Continues without difficulty.

K. 570 3rd movement, bars 1-8
Startup produces correct hypothesis, which ranks tied for 1st.
Continues without difficulty.

K. 576 1st movement, bars 1-8
Startup produces correct hypothesis, which ranks 2nd
Continues without difficulty.

K. 576 2nd movement, bars 1-8
Startup fails.
Continuation loses correct hypothesis in measure 5.

K. 576 3rd movement, bars 1-8
Startup produces correct hypothesis, which ranks 3rd.
Continues without difficulty.

C: Glossary

Ancestor All of the *beat nodes* related to a particular beat node though a *parent* relationship — that is , the beat nodes parent, its parent's parent, and so on — are that beat node's ancestors. An ancestor of a beat node has a larger interval than the beat node, and is depicted with a larger arch.

Beat Levels Beat levels are sequences of *beat nodes* connected by next and prev links.

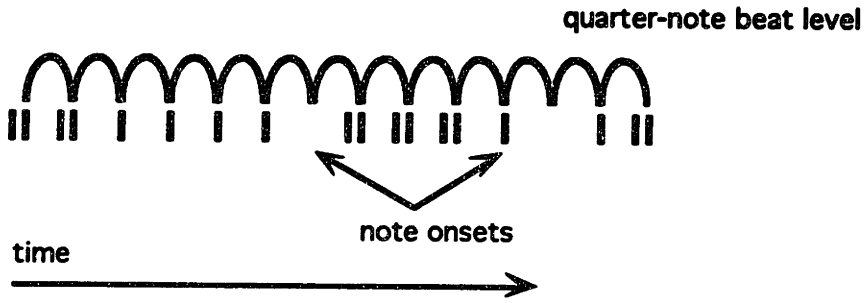


Figure Glos-1. Quarter-note beat level for "La Marseillaise." Note onsets are indicated by the vertical bars under the arches.

A beat level can be thought of as a trace of a listener's finger as it taps to the music. The beat nodes that comprise a single level are normally of the same approximate size; when they are not, it is an indication of a change in tempo.

Beat Nodes Beat nodes correspond to the usual idea of a musical beat. Beat nodes point to *summary events* or *ghost events;* they can be thought of as indications of where a listener might tap. Beat-nodes are embedded in both a list structure and a family hierarchy, so each beat points to its parent, its children, as well as the next and previous beat in a list.

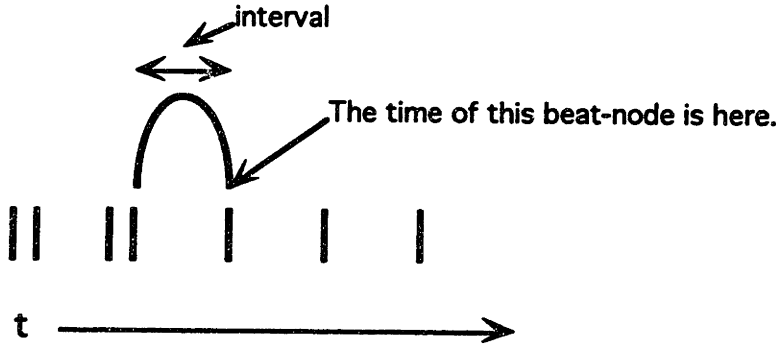


Figure Glos-2. Basic beat node.

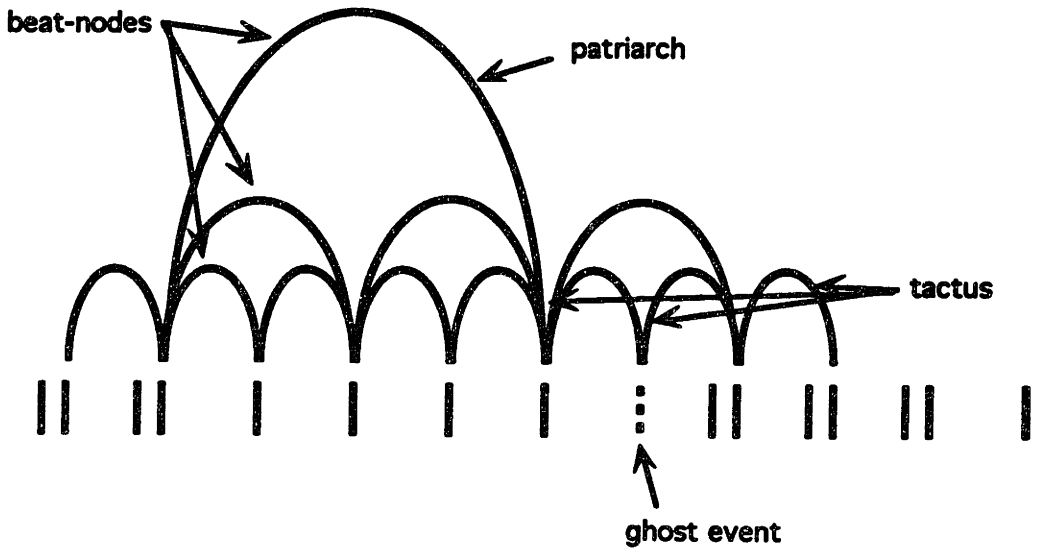


Figure Glos-3.

In appropriate contexts, we speak of beat nodes as having the metrical values from usual musical terminology, i.e., we speak of eighth-note beat nodes, quarter-note beat nodes, half-note beats nodes, and so on. A beat-node's *children* are said to subdivide the beat node; a quarter-note beat node might be subdivided by two eighth-note beat nodes, and may in turn subdivide its *parent*, a half-note beat-node.

Child Two *beat nodes* in a rhythmic hypothesis may be related by a parent-child relationship. This means that time spanned by the beat node's interval is subdivided into smaller spans which comprise the intervals of the beat node's children. In the current version of Machine Rhythm, such subdivisions are assumed to be either binary or ternary, that is, a beat node may have either two or three children.

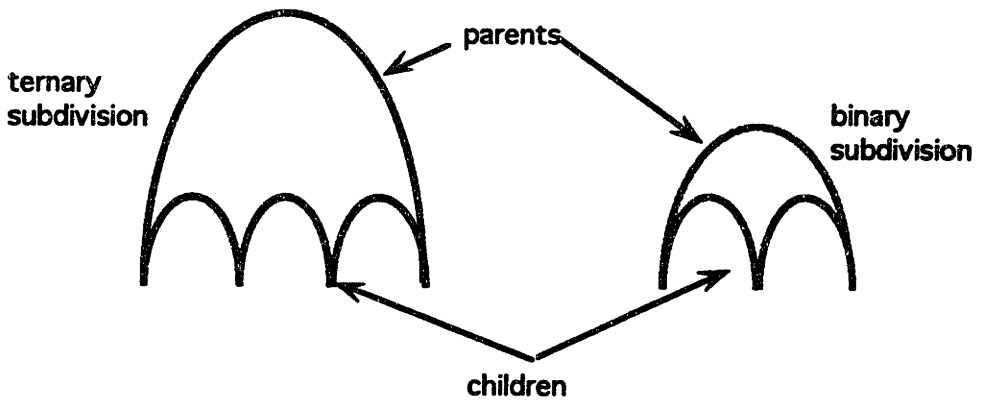


Figure Glos-4. Parent and child beat nodes

Chord A chord is a set of events whose onset-times are effectively simultaneous. Whether or not two events are considered to be part of the same chord depends on the time between their onsets, and their difference in pitch. See the chapter "Preprocessing."

Context A beat node may belong to several different rhythmic hypothesis structures, but it will belong to only one such structure in a given context. Contexts are in one-to-one correspondence with hypotheses. See the chapter "Context-Dependent Links."

Descendent The beat nodes related to a particular beat node through child relationships — that is, its children, its children's children, and so on — are its descendants.

Events An event may generally be thought of as Machine Rhythm's correspondent to a musical note; it is a structure whose attributes are pitch, onset-time, loudness (or velocity, since what is really measured is the velocity with which a key is depressed), duration, and *IOI*, which is the time to the next event. Each event corresponds to two *MIDI* bytes, a note-on and a note-off.

Family A family is the set of *rhythmic hypotheses* that are derived from a common *progenitor.*

Ghost Event A ghost event corresponds to a place in a performance where a listener might tap, but no note sounds. In illustrations in this thesis, ghost events are depicted as dotted lines. (See figure Glos-3.)

Hypotheses See Rhythmic Hypotheses.

Hypothesis Context See Context.

Interval A *beat node's* interval is the time between the onset time of the event that the beat node points to, and the time of the previous beat node's event. The interval is graphically indicated by the width of the arch that depicts the beat node. (See figure Glos-2.)

IOI IOI, or inter-onset-interval, is an attribute of events; it refers to the time between the event's onset-time and the next event's onset time. The IOI of an event depends on the set of events under consideration; it may refer to distance between (regular) events or distance between *summary events.* In a typical situation, the IOI of an event which is a chord-constituent is the time between its onset and the onset of another constituent of the same chord, while the IOI of the chord itself is the time between the chord's onset time and the onset time of the next summary-event.

MIDI MIDI, or Musical Instrument-Digital Interface, is a standard way of recording performance information from electronic instruments. A MIDI representation of a performance is called a *sequence*, and the information available from the sequence usually includes the onset and release times of each note, how loudly the note was played (this parameter is called the velocity, since what is actually measured is how fast a key was depressed), how long it was played, and which pitch it corresponded to. A (musical) note will be represented by two *MIDI bytes*, one for its onset and one for its release.

Offset-Suite The offset-suite of a level is a list of numbers each of which indicates the period of a repeating melodic pattern. An offset-suite of (2 4 8) indicates that there is a repeating two-note pattern embedded in a repeating four-note pattern, which in turn is embedded in a repeating eight-note pattern.

Offset See **Phase**.

Parent Parent is the inverse relation ship of *child;* see the definition above.

Patriarch A *beat node* in a rhythmic hypothesis with no parent is called a patriarch. See figure Glos-3 above.

Performance For our purposes, a performance can be regarded as a list of *MIDI* bytes produced by a musician playing an electronic instrument.

Phase Phase describes a relationship between two levels that are part of the same rhythmic hypothesis, and whose beat-nodes are related through parent-child relationships. The parent-level may be offset by different amounts relative to the child-level; the amount of the offset is the phase. The terms offset and phase are used interchangeably in this thesis.

Figure Glos-5. The two top levels in this figure have a difference phase with respect to the bottom level.

Progenitor A progenitor is a *rhythmic hypothesis* produced by the Startup module; it represents a particular guess about the rhythmic structure of the beginning of the piece. The parsing of the remainder of the performance is accomplished by extending the progenitors. There is generally more than one way of extending a progenitor; the various hypotheses that result from extending a single progenitor are called a *family.*

Rhythmic Hypothesis A rhythmic hypothesis is a set of *beat levels* related by parent-child relationships. It represents a complete description of the rhythmic structure of a performance — its time signature, the length of the upbeat, and the rhythmic role played by each note. See figure Glos-3 above.

Startup Events This is a group of events selected from the beginning of the piece which are used to determine a set of rhythmic hypotheses. These events are meant to correspond to the initial section where the listener first determines the rhythmic structure of the piece.

Summary Events Summary events are either chord or single notes. The list of summary events for a performance can be thought of as a monophonic reduction of a polyphonic performance; the rhythm is determined with respect to the summary events.

Tactus In musical terminology, the tactus is the basic beat of a performance, the beats that a listener would normally tap, or that a conductor would indicate with a baton. In Machine Rhythm, the tactus is the *beat level* which is the lowest level in the rhythmic hypothesis hierarchy; beat nodes that comprise the tactus have no children. The Machine Rhythm tactus may correspond to the musical tactus, or it may correspond to a subdivision of the musical tactus. It might seem natural, for example, to tap or conduct the quarter-notes of a piece, while the tactus used by Machine Rhythm corresponds to eighth-notes.

Treacherous Confusions To Avoid: The Machine Rhythm program uses a number of different structures involving the same sets of objects. I have attempted to use terms that appeal to readers' intuitions and are neither arcane nor confusing; even so, there are a number of possible misunderstandings.

Families are not rhythmic hypotheses. A group of beat nodes related by parent, child, next and prev relation ships is a rhythmic hypothesis. A *set* of rhythmic hypotheses which originate from the same *progenitor* is a family.

Progenitors are not patriarchs. A patriarch is a beat node with no parents; patriarch refers to a beat nodes position in the rhythmic hypothesis structure. A progenitor is a rhythmic hypothesis, from which other rhythmic hypotheses are generated, by the process of *extension.*

D: References

Allen, P., and R. Dannenberg. 1990. "Tracking Musical Beats in Real Time." *Proceedings of the 1990 International Computer Music Conference.* Glasgow: Computer Music Association, pp. 140-143.

Bamberger, J. 1980. "Cognitive Structuring in the Apprehension of Simple Rhythms." *Archives de Psychologie* 48:171-199.

Bartok, B. 1976. *Turkish Folk Music from Asia Minor.* Princeton: Princeton University Press.

Bregman, A. 1990. *Auditory Scene Analysis.* Cambridge: The MIT Press.

Chung, J. 1989. "An Agency for the Perception of Musical Beats." MS Thesis, EECS, Media Laboratory. Cambridge: Massachusetts Institute of Technology.

Desain, P., and H. Honing. 1989. "Quantization of Musical Time: A Connectionist Approach." *Computer Music Journal* 13:56-66.

Ellis, D. 1992. "A Perceptual Representation of Audio." MS Thesis, EECS, Media Laboratory. Cambridge: Massachusetts Institute of Technology.

Fahlman, S. E. 1979. *NETL: A System for Representing and Using Real-World Knowledge.* Cambridge: The MIT Press.

Gold, B. and L. Rabiner. 1969. "Parallel Processing Techniques for Estimating Pitch Periods of Speech in the Time Domain." *J. Acoust. Soc. Am.* 46: 442.

Gordon, J. and E. H. Shortliffe. 1985. "A Method for Managing Evidential Reasoning in a Hierarchical Hypothesis Space." *Artificial Intelligence* 26: 323-357.

Handel, S. 1989. *Listening.* Cambridge: The MIT Press.

Katayose, H., H. Kato, M. Imai, and S. Inokuchi. 1989. "An Approach to an Artificial Music Expert." *Proceedings of the 1989 International Computer Music Conference.* Columbus: Computer Music Association, pp. 139-146.

Jepson, A. and W. Richards. 1991. "What is a Percept?" MIT Center for Cognitive Science Occasional Paper #43.

Lee, C. 1985. "The Rhythmic Interpretation of Simple Musical Sequences: Towards a Perceptual Model." In Howell, Cross & West (eds.), *Musical Structure and Cognition,* pp. 53-69.

Lenat, D. B. and R. V. Guha. 1990. *Building Large Knowledge-Based Systsms:Representation and Inference in the Cyc Project.* New York: Addison-Wesley.

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

Longuet-Higgins, H. C. 1976. "Perception of Melodies." *Nature.* 263:646-653.

Longuet-Higgins, H. C. and C. Lee. 1984. "The Rhythmic Interpretation of Monophonic Music." *Music Perception* 1:424-441.

Longuet-Higgins, H. C. and M. J. Steedman. 1971. "On Interpreting Bach," *Machine Intelligence* 6:221.

Machover, T. and J. Chung. 1988. "Hyperinstruments." MIT Media Lab proposal.

Minsky, M. 1981. " A Framework for Representing Knowledge." In Haugeland (ed.), *Mind Design*, 95-128.

Minsky, M. 1986. *The Society of Mind*. New York: Simon and Schuster.

Mont-Reynaud, B. and M. Goldstein. 1985. "On Finding Rhythmic Patterns in Musical Lines." *Proceedings of the 1985 International Computer Music Conference.* Burnaby, British Columbia: Computer Music Association, pp. 391-397.

Norris, M. and D. Rosenthal. 1990. "The Effect of Varying Loudness and Duration on Human Rhythm Perception." MIT Media Lab Technical Report.

Palmer, C. 1990. Informal communication.

Palmer, C. 1988. "Timing in Skilled Music Performance." Ph.D. thesis, Ithaca: Cornell University.

Povel, D. and P. Essens. 1985. "Perception of Temporal Patterns." *Music Perception* 2:411-440.

Rosenthal, D. 1990. "Computer Emulation of Human Rhythm Perception." MIT Media Lab Technical Report.

Schloss, A. 1985. "On the Automatic Transcription of Percussive Music – from Acoustic Signal to High-Level Analysis." Ph.D. thesis, CCRMA, Department of Music. Palo Alto: Stanford University.

Sloboda, J. 1983. "The Communication of Musical Metre in Piano Performance." *Quarterly Journal of Experimental Psychology* 35A:377-396.

Vercoe, B. 1985. "The Synthetic Performer in the Context of Live Performance." *Proceedings of the 1985 International Computer Music Conference.* Burnaby, British Columbia: Computer Music Association, pp. 25-31.