BoB: An Improvisational Music Companion

Belinda Thom

May 2001

CMU-CS-01-138

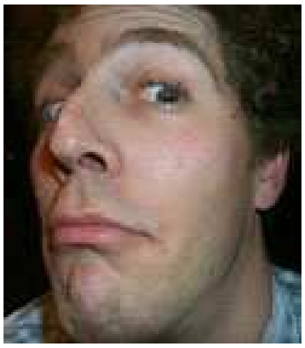
School of Computer Science
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Thesis Committee:
Manuela Veloso, Chair
Roger Dannenberg
Tom Mitchell
David Wessel, University of California, Berkeley

This research was sponsored by the United States Air Force (USAF) Air Force Research Laboratory (AFRL) under agreement no. F30602-98-2-0135 and no. F30602-97-2-0250, the Defense Advanced Projects Research Agency (DARPA) under contract no. N66001-94-C-6037, and generous fellowships from Bell Labs and the National Science Foundation. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the USAF, AFRL, DARPA, NSF, U.S. government or any other entity.

Keywords: Unsupervised Learning, Interactive Computer Music Systems, Music Improvisation, Real-time, Stochastic Processes

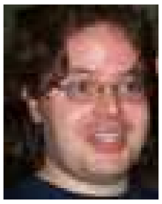
Without them, my personal life would be lived in black-and-white — thanks to Robbie Warner, Jazz, Kiwi, and my friends — who paint in the vibrant Technicolor. Early on, Bill and Carol Thom instilled in me an insatiable love of music; this has brought me great pleasure. Enjoying research and succeeding at CMU was a blast under the guidance of my advisor, Manuela Veloso. Each member of my committee also generously offered their invaluable expertise, which shaped numerous aspects of this interdisciplinary endeavor. Thanks everyone!













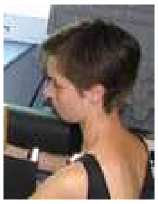










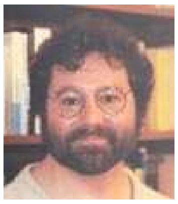








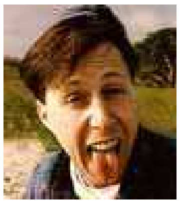


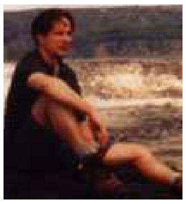
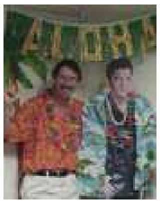




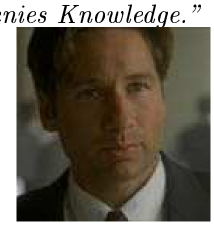
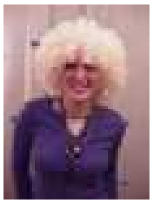


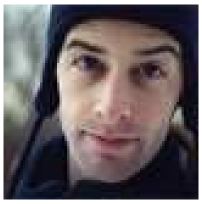




Extreme Secret Agent Kosak: “Government Denies Knowledge.”



Abstract

In this thesis, I introduce a new melody representation scheme and a machine learning framework that enables customized interaction between a live, improvising musician and the computer. The ultimate intent of these technologies is to provide the infrastructure needed to intimately couple the computer with a musician's all-too-transient improvisations; potential applications range from improvisational exploration to education and musical analysis. I introduce Band-OUT-of-a-Box (BoB) — a fully realized agent that trades personalized solos with a simulated user in real-time.

Musical improvisation is an ill-defined situation- and user-specific practice whose inherent non-literal basis makes the authoring techniques exploited in other AI/entertainment systems (e.g., interactive characters or stories) less helpful when building an *improvisational music companion.* A major contribution is BoB's computational melodic improvisation model, which configures itself to its users' unlabeled examples, alleviating some of the burden associated with defining an appropriate musical aesthetic.

I first describe an abstract perception algorithm that maps short strings of notes onto a mixture model. The components of this model correspond to the various *playing modes* — e.g., tonal, intervallic, and directional trends — that the user employed during various parts of a warmup session. I next describe a crucial technology that closes the perception loop by integrating the learned model's parameters into a stochastic process that, when sampled, can produce sequences that exhibit specific abstract goals (or playing modes) while seamlessly integrating into the constraints set up by the local environment. These algorithms' musical performances are evaluated by qualitatively exploring their behavior using two different simulations: both the transcriptions of Bebop saxophonist Charlie Parker and jazz violinist Stephane Grappelli. These algorithm's quantitative performance are also assessed using more traditional machine learning techniques.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis investigates a novel domain for intelligent agents — agents as specific users' improvisational music companions — and introduces *Band-OUT-of-a-Box* (BoB), an agent designed to trade live, customized solos with a specific improvising musician/user. An unsupervised machine learning algorithm is presented that addresses a primary technical challenge in this domain — to computationally specify the agent's musical intelligence, a highly personalized and context dependent concept — by learning to model a user's improvised behavior. This learned model abstracts bars of solos into user-specific playing modes, providing a technology for perceiving what the user plays in a musically powerful, musician-specific manner. In isolation, this model does not provide the agent with the ability to interactively respond to what the user plays. Another crucial technology that this thesis presents is a mechanism for closing the learning loop by reversing the user-specific playing mode abstraction. In particular, the learned model's parameters are incorporated into a stochastic process that, when sampled, produces solos that realize specific user playing modes. With these two technologies, the agent can intelligently respond to the musician's solos in real-time. These ideas are fully implemented in BoB, which is used to simulate and evaluate real-time solo-trading in two contexts: both the improvisations of saxophonist Charlie Parker and jazz violinist Stephane Grappelli.

1.1 Motivation

The vision of my research is to interactively improvise with the computer, using it as a tool when practicing alone to capture and experiment with my all-too-transient spontaneous musical ideas.

1.1.1 Improvisational Music Companions

I want the computer to serve as my own personal *improvisational music companion* (IMC), playing music with me, improvising with me, and getting to know my own unique musical personality in such a way that it could:

> ... answer me halfway through my phrase, bringing me to the point where I can actually sing what its going to play next, and then, instead of playing that, it'll play something against it which compliments what I'm singing in my head.

This quote, which inspires my research, paraphrases drummer Keith Copeland's musings about improvising with jazz pianist Ahmad Jamal.[1] Towards this end, I am creating BoB, an agent that employs novel machine learning techniques to trade solos with its user in a musician- and context-specific manner.

1.1.2 Interactive Music Systems

Most interactive music systems rely on an *author-the-aesthetics* paradigm, meaning that ultimately, the artist, programmer, and/or user is expected to configure the system's inner workings, hand-customizing until interesting musical aesthetics result, e.g., (Dannenberg 1993; Rowe 1993; Pennycook and Stammen 1993).

For example, in many of Dannenberg's interactive performances, the computer is programmed to be an extension of his compositions, capable of responding to the real-time performance decisions and nuances of the human performer. In this setting, music recognition is either hand-coded and/or trained using human supervised learning data, e.g., (Dannenberg, Thom, and Watson 1997), and music generation is generally designed to carry out the composer's, rather than the performer's, goals.

[1] As compiled in (Berliner 1994).

While more open-ended improvisational experiences are sought in Wessel's computer-assisted performance settings, the primary focus is again on human-authored aesthetics. Towards this end, much of his work has revolved around creating technologies for organizing and controlling the access to musical material so that human computer musicians can author meaningful musical experiences on-the-fly (Wessel, Wright, and Kahn 1998; Wessel and Wright 2000).

Situated somewhere between these two approaches is Lewis' interactive *Voyager* system (Casserly 1997; Roads 1985). Improvisationally, *Voyager* is quite open-ended and Lewis points out that the emergent properties that arise "when playing with the thing" are of primary interest to him. This interest coincides more closely with Wessel's agenda than Dannenberg's, which relies more heavily on compositional intent. At the same time, both Dannenberg and Lewis are practicing improvisers that have spent many years hand-crafting their respective interactive software environments.

Some interactive systems have embodied a more autonomous sense of musical aesthetics using *machine learning* (ML) techniques to reduce some of the authoring burden, e.g., (Biles 1998; Franklin 2001). Although learning provides an additional layer of flexibility, because these systems are based on carefully constructed musical representations (in Biles, specialized chord-to-scale based mappings; in Franklin, hand-coded fitness functions), they are not as flexible as they could be. It is also unclear how much of these systems' aesthetically pleasing behavior comes from these carefully constructed representations.

1.1.3 A New Agenda For Musical Intelligence

Improvised companionship in the soloing practice tool context motivates a different research agenda (Thom 2000a). By definition, an IMC is a *specific* musician's *personal companion*; this directly affects how one thinks about the agent and specifies what is important in simulating its musical intelligence. Using machine learning to replace explicit authoring is a primary concern and this technology must be developed to configure a computational improvisation model for a particular arbitrary musician playing at a particular arbitrary time and in a particular arbitrary context. An IMC must provide an extremely flexible and adaptive model of improvised melody that can:

- leverage off of the spontaneous creativity of its user;
- transform what the user plays in interesting new ways;
- reflect the user's performance back to them, that such co-mingling might result in making the user's spontaneity less transient, more tangible and explorable.

Providing an adaptive and robust basis for *reasonably competent* musical behavior is key. This approach is in direct contrast to a more knowledge-intensive one in which a set of specific rules are fine-tuned for a particular user and setting.

It is simply more important to handle a wide variety of improvisational settings reasonably well than it is to expertly handle a specific improvisational genre. For example, I want to be able to use an IMC when improvising on top of:

- the rhythmic, modal types of playing one finds in Ali Farque Toure's guitar or Dr.Dig's didgeridoo music;
- well delineated chord progressions (e.g., blues, bluegrass, folk);
- unaccompanied settings where melodies are completely unconstrained.

To address this need, BoB's architecture is purposefully designed at its most basic level to function in the absence of specific codified jazz or blues theories. **Instead, the focus is to develop a computational model of improvisation that tightly couples with, and fully functions from within the *sole context of*, the user's improvisations.** With respect to computer music, one of BoB's primary distinctions is this unique agenda.

If machine learning is to be relied on, training data collection must be as natural and non-obtrusive as possible to easily accommodate amateur improvisers with varying levels of skill. An IMC cannot require that an expert "twiddle the settings" or expect a user to directly supervise its musical behavior. Furthermore, a machine learning algorithm must be able to learn in the presence of little data because this allows the IMC to learn about the musician's playing style *"right now"*, i.e., in the current solo-trading context. I propose handling smaller datasets by simplifying the underlying mathematical model that is to be learned, the assumption being that a less accurate model trained on more immediately relevant data will provide a more intimate and appropriate interaction with the user than a more complex model that is trained on a larger hodge-podge of the musician's behavior would.

1.2 Band-OUT-of-a-Box

Band-OUT-of-the-Box (BoB) is the IMC contributed by this thesis. BoB's task is to trade live, improvised solos with its user in real time. The BoB experiment explores the idea that a solo-trading companion can be *operationalized* — i.e., algorithmically specified — using a probabilistic melody improvisation model that learns to automatically configure itself to the user's various playing modes after listening to them improvise.

1.2.1 Band-in-a-Box

The name 'BoB' pays tribute to *Band-in-a-Box* (BiB) (PGMusic 1999), a commercial software product that provides fixed accompaniment to musicians while they improvise. While BiB has enhanced many improvisers' at-home practice efforts, it neither listens to nor responds to what the user plays.[2] BiB's success has much to do with the fact that it comes with a huge library of expertly crafted styles, offering a wide range of choice over the type of accompaniment that is provided (like funk, disco, or jazz). Should a user care to build a customized accompaniment style, specific *licks*[3] can be recorded and various parameters can be modified to control how these licks are combined and applied. This product is another example of the author-the-aesthetics paradigm, requiring the user to explicitly configure the system when non-prepackaged musical behaviors are desired.

1.2.2 User-Specific Playing Modes

The ultimate key to success is providing an intimate connection between the musician and BoB — at all times the user must feel listened to and responded to. **The**

[2]Although the new version can improvise by creating solos for a specific harmony and/or making up a new harmony, this "creativity" is non-interactive.

[3]Musicians use the term *lick, riff*, or *motive* to refer to a melodic fragment.

most basic hypothesis in this thesis is that such intimate coupling can be simulated provided the agent is powerful and flexible enough to detect and respond to the various *modes of playing* that the musician employs during different parts of their improvisations. For example, a user's playing mode might amount to their preference for certain tones in a musical scale arranged so that a particular melodic shape and set of intervals is used.

BoB will explore playing modes in terms of how various solo segments utilize different tonal, intervallic, and directional trends. In particular, three different feature vectors for encoding a pitch sequence are considered:

- a *pitch-class*[4] (PC) based view;
- an *intervallic* (INT) based view;
- a *directional* (DIR) based view.

PC usage can be influenced by an underlying *tonality* and/or *musical scale*, so these terms will also be used to refer to various PC-based qualities. Similarly, INT usage, which quantifies the absolute semitone distance between adjacent pitches, will also be referred to in terms of *melodic continuity* or *discontinuity*, which are affected by ones preference for certain *arpeggios*, *triads*, or *runs of notes* within a scale. Finally, DIR usage, which quantifies how a melody's pitches ascend, descend, and/or do not change in pitch over time, will also be referred to in terms of a melody's *contour* or *shape*. Together, these trends are referred to as a *conglomerative* (CGL) view.

1.2.3 Musical Input, Time, and Synchronization

General-purpose music systems that interact with human performers present a number of difficult engineering problems that are the subject of much research e.g., (Winkler 1998; Rowe 1993; Cemgil and Kappen 2001b; Raphael 1999). This thesis ignores three important issues, namely *pitch-tracking*, *tempo-tracking*, and *rhythmic quantization*. Since playing modes are defined in terms of pitch sequence content, a discrete, perceptual notion of *pitch* is used (based on the equal tempered scale) and simple MIDI controllers can be used to input melodic data into the computer.

[4]This use of the word 'class' originates in psychoacoustics and should not be confused with the classes that are inferred by BoB's learning algorithm.

Important terms need to be defined to quantify the passage of time in music, a concept that is crucial if improvisors are to synchronize their playing with one another. Music often has a pulse and its unit of pulse is called a *beat*. Beats are rarely undifferentiated but rather occur in repetitive groups, called *bars*, and these in turn define the music's *meter*. For example, when one taps their foot to a rock-n-roll tune, odds are that they are tapping their foot in '4/4' meter, which amounts to hitting the floor four times per-bar, and probably emphasizing beats one and three. *Tempo* quantifies how quickly the beats pass and is typically measured in terms of beats-per-minute.

With regards to tempo tracking, it is assumed that the user will be able to "keep up" with BoB's notion of the beat, synchronizing themselves to a non-interactive computer accompaniment that BoB provides (a similar scheme was successfully employed in BiB). Rhythmic quantization will be the first thing addressed in future work because, without it, live interaction with a user is not possible. **Currently, "live" interaction is simulated in real-time using human transcribed improvisations.**

1.2.4 Interactive Solo Trading

In BoB, interactivity is staged in terms of *call* and *response*, mediated by a *lead sheet*. The lead sheet specifies high-level aspects of the interaction — what tempo and harmony or rhythm background to use, in what bars each player is expected to solo, and so on. This information is specified by the user before interaction begins.

Consider *trading fours*, a solo trading interaction in which the following is repeated over and over:

1. Soloist A plays a call, improvising for four bars while soloist B listens.

2. Soloist B improvises a response to this call while soloist A listens.

In reality, the distinction between call and response is blurred because soloist B's response can be viewed as a call for soloist A.

Figure 1.1 provides one example, displaying a different staff for each soloist. The musician has just played their first four bars of solo (top staff). BoB is replying (bottom staff), having just performed its first two bars of response, now scheduling its third, and creating its fourth. The accompanist's part is listed underneath the bottom staff in standard chord notation. This part is played non-interactively.

Figure 1.1: BoB and the Musician Trade Fours

1.2.5 High-Level Architecture

BoB is designed to handle two soloists: itself and the user, only one of whom may improvise at a given time. In addition, each soloist must play a *monophonic* melody, meaning that they only play one pitch at a time. In Figure 1.2, BoB's architecture is outlined.

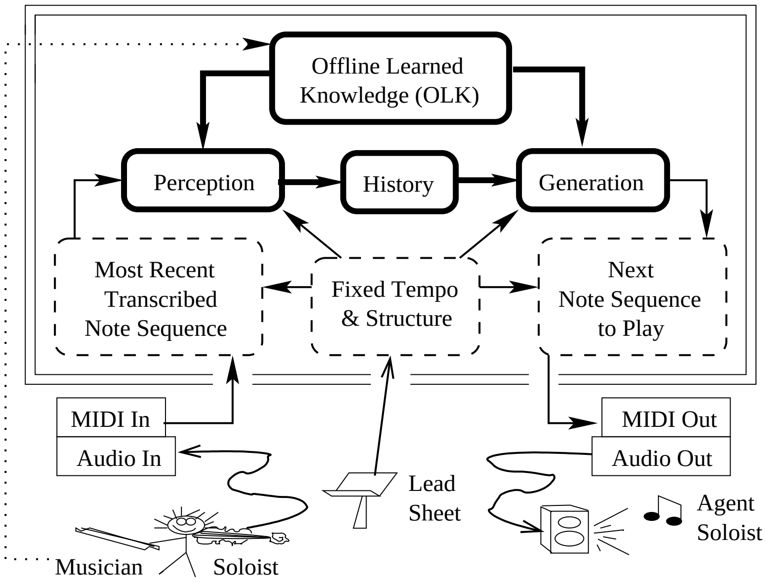


Figure 1.2: The Band-OUT-of-a-Box Architecture

The *environment* consists of everything that lies outside of BoB (the double-lined box) and includes the musician, the lead sheet, and off-the-shelf MIDI and audio interfaces. Inside of BoB, real-time MIDI and event processing are handled by the dashed-box components. This thesis addresses the bold components, which include:

1. perception;
2. generation;
3. history;
4. off-line learned knowledge;

and are entirely responsible for BoB's interactive, musically intelligent behavior.

The *perception* component perceives the musician's solo call on a *per-bar* basis, abstracting each bar into an appropriate user playing mode. As the musician's call progresses, a sequence of these abstractions is stored by the *history* component. History is in turn used by the *generation* component to guide BoB's search for a user- and context-specific response. The dotted-line connecting the musician to the *off-line learned knowledge* component (OLK) indicates that user knowledge was learned before interactive solo-trading began.

OLK is configured as follows. First, BoB collects training data by transforming each bar that the user improvises into a conglomerative view. This data collection is referred to as the *warmup session*. Next, this data is taken off-line and a model is learned for mapping conglomerative views into distinct user playing modes. When BoB goes back online, interactively trading solos with the user in real-time, this model is used to operationalize both user- and context-specific solo perception and generation.

1.2.6 Learning a Model of the User

A user-specific *training set* is created by breaking the warmup data into local segments, one for each bar of solo. Each segment is transformed into a conglomerative view. Learning involves probabilistically *clustering* or *partitioning* these views into a specific number of *components* or *classes* so that they appear to be maximally likely. This clustering is done to capture the different playing modes that the musician uses when spontaneously constructing solos. Since it is not known which bars belong to what classes, to learn this clustering function is an *unsupervised learning* (UL) problem. The goal is not to best predict a target function, but rather to "best" explain the features contained within the training data.

A *mixture-model* based approach is used, providing both a mechanism for perception (estimate what class is most likely to have generated a given view) and generation (derive a new solo from a specific class by sampling from an appropriate distribution). This probabilistic model also comprehends likelihood, which can be used to interpret how *surprising* a particular perception appears. An important distinction of this approach is that the user's warmup data itself determines how different CGL views will be perceived as "similar." In addition, similarity is derived from the data in a very open-ended, flexible, and context-sensitive way.

1.3 Contributions

As highlighted below, this thesis makes five contributions to the fields of machine learning, intelligent agents, and interactive music systems.

1.3.1 The Improvisational Music Companion

This thesis formally introduces agents as spontaneous musical companions. The focus is on a tight, intimate coupling between the improviser and his or her machine/companion, forcing user-customization, adaptation, and musical sensibility into the limelight. Just as believable agents presented a new paradigm for artificial intelligence (AI), IMCs present a new paradigm for intelligent music systems. The goal shifts from engineering an agent with specific musical expertise to providing a flexible, adaptive, common-sense modicum of musical behavior.

1.3.2 A Hierarchical Set of Melodic Views

This thesis develops a methodology for automatically decomposing a monophonic solo into a temporal sequence of local segments. The term 'segment' instead of 'bar' highlights that, although in this thesis, segmentation is per-bar, the technology can handle arbitrary segmentation schemes.

Each segment is encoded in multiple ways, including:

1. a transcribed string-of-notes;

2. a hierarchal rhythm tree;

3. a pitch sequence;

4. a conglomerative viewpoint;

5. a user-specific playing mode.

This list is ordered because it is hierarchical. Each successive encoding produces a more abstract view of the melody than its predecessors.

1.3.3 An IMC Solo-Trading Agent Architecture

The BoB architecture contributes the following technologies:

- a real-time, probabilistic human/computer solo-trading model whose feature space and abstraction mechanisms are simple and musically meaningful, making the model easy to understand, reason about, and extend.
- a learning mechanism that configures the model to the musician's playing modes after merely listening to him or her improvise.
- an integrated procedure for generating new solos that maintain a particular set of user-specific abstractions and seamlessly integrate into the evolving musical environment, domain knowledge, etc.

1.3.4 An Algorithm for User-Specific Perception

A mixture model composed of *Multinomial* probability distribution components is introduced for clustering histograms that contain an arbitrary number of counts. This model is derived and implemented so that:

- The total number of counts in a histogram is a random variable.
- All of the counts within a histogram are modeled as belonging to the same component.
- The learning algorithm can infer a useful model even when histogram data is relatively *sparse*, i.e., few counts are distributed among a histogram's many bins and training data consists of a limited number of histograms.

This technology is used to customize BoB's solo-trading model to the user, partitioning the CGL feature space into distinct playing modes. This model is tested on a variety of datasets, both simulated and musician-specific, including some Charlie Parker and Stephane Grappelli transcriptions.

1.3.5 An Algorithm for "In Kind" Response

An "in kind" response to a solo call is obtained in two stages. First, a stochastic algorithm is introduced to modify the fringe of a call rhythm's tree while maintaining important invariants in the tree's internal structure. A pitch-sequence generation algorithm that closes the learning loop is derived to fill in the pitch content of the tree's leaves. This algorithm is *goal driven* in that it can produce sequences that map back into specific user playing modes. This algorithm also maintains a fairly arbitrary set of constraints and can be used, for example, to generate a pitch sequence that: 1) starts and ends on specific tones, and 2) "fills in the blanks along the way" in such a way that the goal's tonal, intervallic, and directional trends emerge. This algorithm relies on a heuristic that efficiently integrates what was learned in the user-model into a Markov chain sampling scheme. This model is tested by simulating a variety of real-time solo trading interactions using Charlie Parker's and Stephane Grappelli's transcriptions.

1.4 A Reader's Guide to the Thesis

What follows is a general description of this thesis' contents, including a brief summary of each chapter and some guidelines for interpreting mathematical notation.

1.4.1 Outline

- **Chapter 2** situates the IMC task within the jazz and blues tradition. The foundational aspirations of a computational improvisation model are described. Various aspects of the IMC domain are fleshed out and situated within AI.

- **Chapter 3** expands upon the *Band-OUT-of-a-Box* architecture presented in Section 1.2. Handling real-time melodic input is addressed in both current (simulated live) and future (live musician) paradigms. How BoB's intelligent music components operate upon and transform this input and interact with one another and the musician are also presented.

- **Chapter 4** introduces the two most basic data structures that encode melodic segments. A tree-based structure delineates rhythm and is used to produce the histogram-based structures that encode a segment's conglomerative pitch-sequence trends.

- **Chapter 5** introduces the algorithm that learns to map conglomerative histograms into abstract, user-specific playing modes. So as not to detract from its usefulness in other discrete, temporal domains, this algorithm's derivation is general. Specific issues concerning the model's application to BoB are only presented when necessary.
- **Chapter 6** introduces the two inter-related stochastic algorithms that are used to generate an "in kind" response to a solo call. Rhythmic transformation is tree-based, capable of permuting a call's rhythm in musically reasonable ways. Goal-driven pitch sequence generation, which is tightly coupled to the learned user-model, assigns new pitches to this rhythm.
- **Chapter 7** assesses the performance of BoB's customized perception and goal-driven generation from a musical point of view. The former is assessed qualitatively and the latter in terms of listening experiments. This chapter is crucial because it validates that: 1) the ML approach in Chapter 5 infers salient musical structure, and 2) the heuristic approach in Chapter 6 generates solos that exhibit this musical structure.
- **Chapter 8** describes related work, limitations of BoB's current implementation, and promising directions for future research.
- **Chapter 9** concludes, summarizing the long-term vision that motivates this thesis and the contributions that it makes to the fields of machine learning, intelligent agents, and interactive music systems.

The appendices that accompany the representation (Appendix C), learning (Appendix D), and music evaluation (Appendices E through H) chapters are referred to as needed in the main text.

1.4.2 Notation

In addition to the mathematical nomenclature presented below, readers should familiarize themselves with the appendices that outline pseudo-code (Appendix A) and musical score (Appendix B) syntax.

Italicized Variables

Most variables are italicized, e.g. x, y, and z. The following names are reserved for specific contexts:

1. s is a transcribed segment, and v and x are its corresponding rhythm tree and histogram.

2. Histogram x contains a total of sz counts.

3. Data-point x belongs to dataset X.

4. y is the classification of x, z and ψ are its corresponding indicator and posterior vectors (defined in Sections 5.4.1 and 5.4.2), and l and u are its corresponding labeled and unlabeled likelihoods (defined in Section 5.4.1). Y and Z contain the corresponding values for each data-point in X. L and U are entire dataset likelihoods.

5. Probability distribution G is parameterized by Ω, which might contain probability vectors θ, π, and/or η. When sampling a single data-point from this distribution, notation $x \sim \mathrm{G}(\Omega)$ is used. When independently sampling an entire dataset, notation $X \overset{\mathsf{N}}{\sim} \mathrm{G}(\Omega)$ is used.

Rather than using longer variable names, whenever possible, descriptive subscripts provide additional information, e.g., x_{max} or x_{min}. Descriptive subscripts appear in italicized font.

Variables With Special Fonts

Non-italicized variables are used in the following situations:

- Pseudo-code functions are identified by the `Terminal` type font.

- Probabilistic functions, e.g., $\Pr(x)$, and musical pitches, e.g., B or D♭, are identified by the Roman type font.

- Except for the italicized sets already mentioned, sets are identified by calligraphic letters, e.g., $\mathcal{S}$. The number of items a set contains is correspondingly notated, e.g. $n_{\mathcal{S}}$. Operator # returns the number of items in a set.

Indexing Variables

When variables appear in San Serif type font, they refer to a particular index within a set, sequence, or vector. A variable's name implies its meaning:

- n refers to a particular data-point in a fixed-size dataset. When iterating over index n, the range is $1 \leq \mathsf{n} \leq \mathsf{N}$.

- m refers to a specific dimension in a fixed-size vector. The corresponding range is $1 \leq \mathsf{m} \leq \mathsf{M}$.
- k refers to a specific class within a set of components. The corresponding range is $1 \leq \mathsf{k} \leq \mathsf{K}$.
- Since s refers to a specific item in set $\mathcal{S}$, its range is $1 \leq \mathsf{s} \leq n_{\mathcal{S}}$.
- r refers to an arbitrary real-time segment.
- t refers to a particular element in an arbitrary-length sequence. For example, supposing the sequence corresponds to bar r, then the range would be $1 \leq \mathsf{t} \leq \mathsf{T}_{\mathsf{r}}$.[5]

Special Superscripts

When a variable is superscripted by one (or both) of the following fonts, certain information is implied. The first ***font type*** indicates what training set a variable is based on. For example:

- $X^{\boldsymbol{P}}$ refers to the Charlie Parker training set, and so $\Omega^{\boldsymbol{P}}$ refers to the model learned using this data, $Y^{\boldsymbol{P}}$ contains the results of using this model to classify the training set, etc.
- $X^{\boldsymbol{G}}$, $\Omega^{\boldsymbol{G}}$, $Y^{\boldsymbol{G}}$, etc., are similarly defined for Stephane Grappelli's data.

The second FONT TYPE indicates what part of the conglomerative feature set is being used. For example:

- X^{P} is a training set's PC data.
- X^{I} is a training set's INT data.
- X^{D} is a training set's DIR data.
- X^{C} is the the training set composed out of all of the above.

[5] This distinction is needed so that sequence lengths can vary from segment to segment.

Chapter 2

Improvisational Music Companions

This chapter introduces the improvisational music companion (IMC) domain in more detail. To begin, the solo-trading task is situated within the jazz and blues tradition. From within this context, the foundational aspirations of my computational melodic improvisation model are described. With these foundations in place, various aspects of the IMC domain are fleshed out and situated within the AI community. I first consider how believable-agent methodologies contribute to IMC agents and identify ways in which the IMC domain forces one to extend and modify this methodology. Next, I examine how machine learning is typically used to aid in operationalizing musically intelligent behavior and identify how IMCs force one to significantly alter this paradigm.

2.1 Jazz and Blues Improvisation

The jazz or blues *jam session*, where during which musicians trade solos and play off of one other over a fixed harmonic background, is a useful context in which to think about BoB. Because jazz and blues are long-standing cultural traditions, when evaluating BoB's performance from within these contexts, care must be taken not to let ones expectations about genre get in the way of objectively appreciating all that BoB can do.

For example, in the jazz context, a listener who is asked to compare a specific musician's solo with a BoB-generated solo response might observe that "BoB's response does not complement the musician's solo as a jazz soloist would." When asked to define the notion of complement, an answer like "there should be some sort of contrast or shift in emotion" might come up.

While complementing is an important musical skill, it is a fairly abstract and intentional notion. I believe that this type of behavior is better served later on, after BoB's more instantaneous and immediate musical perception and generation mechanisms are firmly in place. Similar arguments can be made for other abstract intentional notions, e.g., emotionally coloring a performance's realization (Bresin and Friberg 2000) or communicating high-level notions like "lyrical" or "bluesy" (Dannenberg, Thom, and Watson 1997).

2.2 Modeling Melodic Improvisation

BoB's most powerful technologies include its:

1. open-endedness and flexibility;
2. automated user- and context-specific model;
3. constrained stochasticity.

My current focus is to provide a highly constrained, self-configuring, stochastic mechanism by which the computer can spontaneously interact with the musician on a very basic level. Rather than relying on rule-based, symbolic reasoning, this basic, more *primitive* technology generates short segments of notes using a self-configuring, distributed mechanism that, when sampled, can produce note sequences that exhibit specific abstract goals and integrate into the context of the evolving musical exchange. I consciously chose this approach because I believe that it captures the essence of improvisation:

> The ability to improvise [...] seems to come out of the end of ones fingertips [...] its main components are profoundly unconscious. (Sudnow 1978)

My approach whole-heartedly embraces the psychological theory of Johnson-Laird (1991), which opposes the view that jazz improvisation can be adequately modeled by rearrangement and simple transformation of pre-acquired licks so as to meet the exigencies of the current harmony. Johnson-Laird (1991) also makes a convincing argument concerning the computational plausibility of a stochastic model that is based on a greedy search through a highly constrained "what to play next" space. This argument leads to the important question of from where such constraints would come. In my experience improvising on the violin, the more I practice, the more things just start to "come out of the ends of my fingertips." As these physical motions sink in, my ability to generalize them to different settings comes naturally. My constraints have somehow emerged from a lengthy period of committed practice, genre immersion, and so on. BoB will derive its constraints from the user via the learning algorithm and the environment.

2.3 Believable Agents

Just as *believable agents* (BA) presented a new paradigm to the AI community (Mateas 1999), IMCs present a new paradigm for intelligent music systems. Important parallels can be drawn between BAs and IMCs and such comparison is valuable because it:

1. provides deeper insight into the nature of the IMC domain;
2. identifies BA-centric perspectives that naturally simplify the task of building an IMC;
3. identifies important differences between traditional BAs and IMCs, forcing existing paradigms to be extended and/or reinvented.

In both domains the primary concern is resolving the inherent conflict that arises when an intelligent system and an artist interact, the key issue of the conflict being to find the right balance between artistic control (authoring) and autonomous intelligence. For an IMC, this amounts to automating musically-appropriate behavior.

The concept of a believable agent, originally developed by the *Oz Project* at Carnegie Mellon University (Bates 1992), explores building interactive virtual worlds whose characters and story lines are believable. While the nature of believable agents is debated, a central theme is creating interactive, autonomous agents that exhibit rich and compelling personalities (Elliot and Brzezinski 1998; Frank, Stern, and Resner 1997; Mateas 1999). A fundamental reason why believable agents stimulate an appropriate way of thinking about IMCs is their audience- and artist-centered philosophy, whereby:

> [...] building a new kind of cultural experience: engaging, compelling, and hopefully beautiful and profound [...] (Mateas 1999)

is embraced. **Of enormous benefit here is the belief that artistic goals are at least as valuable as purely technical ones, that good art (transformed into a computational setting) breeds good science, but that the converse is not necessarily true** (Mateas 2001a).

2.3.1 Useful Methodologies

In creating autonomous interactive characters with the same rich personalities as those created by artists, BA researchers have attempted to understand the reflections of practicing character artists and to integrate these insights into their methodology. Not surprisingly, the reflections of practicing improvisers have proven similarly useful in my research, e.g., (Bailey 1992; Berliner 1994; Sudnow 1978; Johnson-Laird 1991).

How inherently artistic pursuits influence BA methodology is instructive. Because there is an emphasis on building interactive characters, an important goal is to provide character artists with the same level of artistic control that, for example, animators have. In BoB, a similar goal is pursued: to tightly couple the agent and improviser so that an intimate musical exchange can take place while providing the agent with enough freedom to generate novel material.

Useful believable agent methodologies include:

1. **Stress specificity.** While traditional AI systems seek to capture the general theory of some domain, an IMC must capture the notable aspects of its user's improvisations. Bailey (1992) highlights this point:

   > There is no general widely held theory of improvisation and I would have thought it self-evident that improvisation has no existence outside of its practice.

   Typically, computational jazz systems have been rule-based, instantiating some combination of textbook theories about jazz, (Pennycook and Stammen 1993; Levitt 1981; Russell 1959; Coker 1970), making this methodology a significant point of departure.

2. **Measure success in terms of audience perception.** Classical AI systems seek to optimize some form of objective measure. An IMC must first be built and then evaluated in terms of how engaging its interactions with the user are.

3. **Have strong affinities with behavioral AI.** In particular, an IMC should prefer the broad, shallow capture of musical capabilities. Fortunately, this insight makes the task of building an IMC more doable. While a more traditional AI-based music approach would focus on realizing a musical expert, an IMC merely needs to be able to provide a modicum of customized musical appropriateness in a variety of improvisational settings.

4. **Assume the audience is willing to suspend their disbelief.** This insight is perhaps the most important one because it directly affects how to think about the user (Sengers 1998). With BoB, it is safe to assume that the musician *wants* to use it to share an engaging musical exchange. As such, the user will be more likely to overlook BoB's shortcomings (such as performing an otherwise interesting solo mechanically). A measure of BiB's success is owed to this fact — many of its accompaniment styles are somewhat canned, yet the experience of playing with a "cheesy" band versus playing with no band at all is a powerful one.

2.3.2 Points of Departure

Believable agents research focuses on character, personality, and story — expressive mediums that heavily rely on language-based, literal reasoning. As a result, many BA technologies do not provide viable solutions for controlling the spontaneous generation of musician- and context-specific melodic response.

In an interview with Bailey (1992), jazz saxophonist Steve Lacy noted that:

> The difference between composition and improvisation is that in composition you have all the time you want to decide what to say in fifteen seconds, while in improvisation you have exactly fifteen seconds.

This observation prompts me to question the usefulness of authoring in IMC domains. With BAs, the specific details of an interactive story or character may be decided in the last fifteen seconds, however the ease in which they can be literally expressed and concretely reasoned about provides an opportunity for in-depth specification. As such, the focus on *authorship* — e.g., providing artists with a set of abstractions, tools, and programming language constructs that allow them to create the interactive, artistic experience they envision — is a key part of BA-related research, e.g., (Mateas 2001b; Loyall 1997; Sengers 1998).

BA researchers could technically address improvised story telling (Weyrauch 1997; Hayes-Roth and van Gent 1997; Mateas 2001b) because significant amounts of information can be reasoned about and authored within a reactive programming language like *Hap* (Loyall 1997). For example, in *Oz* a character artist could specify that the shy *Woggle* always quivers when frightened, except when their fear is larger than some internally specified amount; in that case they become paralyzed. Operationally useful aspects of these behaviors can also be written down: shake a certain number of cycles per minute, enlarge the eyes by a certain percentage when paralyzed, etc.

In improvised musical expression, these more tangible aspects of character and personality break down. Although musicians do speak of personality, they usually do so in frustratingly vague terms. Imagine trying to operationalize jazzist Ronnie Scott's notion of such a concept:

> "I would like ideally to express my, I don't know, personality or whatever, musically, to the limits of my ability [.... to ...] play in such a way that it would be recognizable as me, and it would express something to people about the way I feel about things." (Bailey 1992)

In an IMC, it makes sense to author those things that can be easily specified in advance (like tempo) via the lead sheet abstraction. The crucial challenge is to provide a mechanism that can operationalize the construction of spontaneous context- and musician-specific melody. It is simply unreasonable to expect a musician to operationalize details concerning what note sequence to play next. Bailey (1992), who interviewed an extensive range of a practicing improvisers in his book (Bailey 1992), sums it up best:

> [...] The improvisers I spoke to, in almost all cases, did not find any sort of technical description adequate. It wasn't that they weren't interested in technical matters. They just did not find them suitable for illuminating improvisation. They finally chose to describe it in so-called 'abstract' terms. And it became clear that, whatever its deficiencies, this is the best method available.

Whether or not a usable ontology for IMC authoring *could* exist, we certainly do not have the ability to specify such behaviors in operational terms at the moment. However, it is imperative that BoB understand the musical abstractions in which its user speaks, develop an operational musical vocabulary that allows it to abstractly perceive what the musician is doing, and respond accordingly. My strategy for dealing with this is to rely more heavily on machine learning (ML) than is traditionally done when dealing with character and story-based mediums to reduce the authoring burden. This contrasts with the philosophy adopted by some BA researchers who discount ML as undesirable because it might hinder, or automate away aspects of, character and plot that the author explicitly wishes to control.

Another distinction between IMCs and BAs is that a BA is typically authored by a team of experts (character artists, graphic artists) and the audience that experiences their world is a completely separate entity. With IMCs, the improviser is both the audience and the author of the experience. In this context 'author' does not imply one who uses some authoring tools for an IMC's configuration. Rather the word reflects the fact that the musician's own improvisations contain within them the essence of what constitutes musically appropriate behavior.

Another difference is how BAs and IMCs can be evaluated. It is not clear that an improvisational musical experience can be critiqued outside of the physical context within which it was played (Bailey 1992; Sudnow 1978). For example, while it makes sense to focus on a user's engagement with a personality post-facto — much in the same way that it makes sense to ask a viewer to summarize a movie's plot after experiencing it — it is not clear that anything but an improvisor's transitory, immediate musical experience is worthy of attention. On this point, organist Stephen Hicks notes that, with respect to the evaluation of his improvisations:

> "It's either good or bad but if you listen to an improvisation over and over again, it just gets worse. [...] It's something that should be heard, enjoyed or otherwise, and then completely forgotten." (Bailey 1992)

One way in which I might address this issue in the future is to monitor a musician's feedback on-the-fly. For instance, for this purpose, Franklin (2001) proposes using real-time facial expression detectors.

2.4 Machine Learning & Intelligent Music Systems

As we transcribe and analyze music in symbolic terms, it may seem natural to apply knowledge-based methods to an IMCs development. However, spontaneously creating melody, as simple an act as improvised humming, is fundamentally as much about ingenious use of exceptions as it is about adherence to a set of rules (Loy 1991; Papadopoulos and Wiggins 1999). For this reason, in BoB, a primitive technology that generates short segments of notes using a probabilistic learning model whose structure is distributed among various components and feature vectors is pursued. **At the same time, the user model that is built was carefully designed to make it easy to understand and reason about musically; this will allow higher-level musical knowledge and/or user desires to be incorporated later.**

Because of the difficulties involved in hand-coding musical behavior, ML techniques are commonly used when engineering an intelligent music system. Typically, one or more of the following tenets are embraced when applying ML to music:

- **Tenet I:** Use supervised learning to predict the next note (chord, musical event, etc.) in a composed and/or performed musical sequence, e.g., (Hild, Feulner, and Menzel 1991; Feulner and Hörnel 1994; Thom 1995; Widmer 1996; Reis 1999; Weigend and Gershenfeld 1993);

- **Tenet II:** Base learning on a large musical corpus, for example, sets of jazz tunes, J.S. Bach chorales, etc., e.g., (Hild, Feulner, and Menzel 1991; Feulner and Hörnel 1994; Thom 1995; Widmer 1996; Reis 1999; Cope 1992; Weigend and Gershenfeld 1993; Bellgard 1993);

- **Tenet III:** Explicitly incorporate higher-level features into the representation, for example (Feulner and Hörnel 1994; Hörnel and Menzel 1998; Rolland and Ganascia 1996; Cope 1992) all employ features that rely on human-derived harmonic functionality.

I will now argue why, for an IMC, these tenets are not necessarily appropriate.

2.4.1 Predictive Learning

In (Thom 1995), I tried to learn a mapping from a recent sequence of chords into the next chord in a harmonic jazz sequence. A key issue in this type of learning scenario is what cost to use when penalizing a prediction that does not match its target value in the training set. I (as well as many others) had used zero-one cost, meaning that, in (Thom 1995), all chord values except the target were penalized equally. Improvisers, however, sensibly report that the appropriateness of a specific chord is not all-or-nothing, but depends on many things, including: harmonic and melodic context, evolving trends in tension and relaxation, etc.

Although a probabilistic model and cross-entropy cost function addresses the all-or-nothing problem (Bishop 1995; Hörnel and Menzel 1998), it does not address the fact that each chord does not exist in isolation but rather is situated within, and inspired by, a surrounding local chunk of chords. This argument also applies to melody, where musical thoughts are composed of licks, phrases, motives, etc. Ultimately, this line of thinking leads me to question the suitability of using a string-of-events based prediction scheme to simulate the creative behavior required of an IMC.

With interactive improvisation, the goal is to "listen to me, but not to play my stuff back at me"[1] — to do something a little bit different, which can range from transforming a situation into something unexpected to bringing it back to the norm. Novelty, outliers, and average behaviors are equally important, yet prediction-based paradigms will explicitly attempt to memorize their training sequences and, failing that, estimate sequences that best describe the data's average sequential behavior. While all learning schemes attempt to balance generalization and memorization needs, **I fear that "predicting-the-next-thing" contains a more myopic, anti-creative bias than a less rigid approach.**

2.4.2 Large Training Sets

The main reason to train with a larger corpus is to minimize the chance of *over-fitting*, which results when there is not enough data upon which to base inference. While this approach has been useful in learning to predict if something is in the style of, for example, J.S. Bach (Hörnel and Menzel 1998), with an IMC this notion of style breaks down. For example, seasoned improvisors have told me that a musician might play very differently at different times. Furthermore, the causes of this variation are no doubt diverse and fickle (what did they eat for dinner that night? who are they are listening to these days? etc.).

[1] Jazzist John McNeil, as quoted in Berliner (1994).

Interestingly, recent research has reflected upon the benefits of per-song based training, e.g., (Thom 1995; Widmer 1996; Cunha and Romalho 2000). An important issue seems to be that, although there are underlying trends to identify particular genres, players, and composers, much of what makes a song cohesive is the structure found *within itself.*

By collecting data in a preliminary warmup session, as was done in (Dannenberg, Thom, and Watson 1997), I hope to capture the musician's most recent state-of-mind. The primary disadvantage of this approach is that a training set's size is now inherently limited. This issue will be addressed by learning a simpler model. **The assumption is that inference with a smaller, more relevant set of data will compensate for the model's simplifying assumptions.** Ultimately, an IMC's learning task shifts from one of learning a broad-based, stylistic notion of its user to a much finer one — to in essence learn their playing style "right now."

2.4.3 Higher-Level Harmonic Knowledge

Clearly, a system's ability to generalize will be improved when pertinent higher-level features are included because the system no longer needs to learn them from a more basic set of features. However, when the musical common sense contained within these features are obtained from a human expert, these features cannot necessarily be used in an IMC.

Consider the significant amount of musical common sense that is automatically included when an improvisational melody learning system is told what chords were transcribed to accompany a soloist's improvisation. In improvised settings, this knowledge is especially important because chord substitutions abound, roots are often missing, color-tones are common, etc. With IMCs it is not even clear when a quintessential transcription of an accompaniment exists. With respect to the chords in Parker's *Omnibook*, a seasoned improviser once told me "...oh, we always used to change those."

It can be safely argued that BoB would exhibit more musical aptitude if, for example, when a 12-bar blues progression in the key of B♭ was being used as the basis of accompaniment, the learning algorithm was made explicitly aware of this fact (for more discussion, see Section 3.2.3). With a lead-sheet that contains harmonic information and a computer-provided tempo, such knowledge would not even require real-time chord transcription because BoB could simply look up the chords. Why then do I explicitly avoid providing such knowledge to BoB's learning algorithm?

One reason is my primitive melodic focus. I want BoB to spontaneously generate solos in arbitrary settings, including situations where harmonic structure does not exist or is not so important (e.g., when improvisations are more vertical or modally based, when melodic improvisation takes place on top of a tabla drum or other rhythmic backdrop, etc.). **I also believe that there is great value in relying first and foremost on the content *within* the musician's improvisations when inferring salient tonal features about them.** Certainly, such an approach can only be more flexible, adaptive, and self contained. On the other hand, because harmonic knowledge is such an important part of much improvised music-making, care is taken to ensure that BoB's model lends itself naturally to being extended so that harmonic knowledge can be incorporated, when appropriate, at a more abstract level (Section 6.3.5).

Chapter 3

The BoB Solo-Trading Architecture

A high-level view of BoB's musically intelligent components was presented in Chapter 1 (Figure 1.2). This chapter begins by discussing the transformation of a musician's live audio into an input that BoB can use. An important assumption is that BoB receives *transcribed* input, and since the automation of this task is beyond this thesis' scope, "live" interaction is simulated in real-time using human-transcribed improvisations. How BoB's intelligent music components operate upon and transform this input and interact with one another and the musician are also presented.

3.1 User Input

Live improvisers produce audio signals that the computer needs to understand. This is a very difficult problem and a subject of active research, e.g., (Winkler 1998; Rowe 1993; Cemgil and Kappen 2001b; Raphael 1999). At the least, BoB needs to convert audio into a sequence of pitches that can be aligned on top of the music's underlying beat in a meaningful way.

It is assumed that off-the-shelf MIDI devices and/or software can acceptably perform the audio to pitch sequence conversion. Rhythmic alignment involves both tempo tracking and rhythm quantization, and this topic is a Ph.D. thesis in and of itself.[1] Tempo-tracking can be avoided by requiring that the musician synchronize

[1] This subject is the topic of A.T. Cemgil's current thesis work.

themselves to the agent's notion of beat. Rhythmic quantization is the serious remaining issue, and this will be the first thing addressed in future work. Without this functionality, BoB is cannot parse a live musician's solos, which is a necessary precursor to live interaction. Currently, interaction is simulated in real-time using human transcriptions of improvisations whose rhythms are known to parse in advance. This simulated real-time scheme, and the ways in which it will be extended to accommodate live performance, are now described.

3.1.1 Raw Solo Input

BoB can already buffer a stream of MIDI (Association 1989) solo data in real-time, but it cannot yet convert this data into a format that is guaranteed to parse into the tree structure required of the perception component.

Although transforming audio into MIDI can be difficult and error-prone, especially for an instrument like the violin, e.g., (Yoo and Fujinaga 1999), reasonable success has been reported for acoustic instruments like voice and flute using the *Sound2Midi* software (AudioWorks 1999; Franklin 2001). I assume that such a technology can provide BoB with a live stream of MIDI `note-on` and `note-off` events and that this technology will be able to handle a reasonable range of performance situations.

When BoB buffers MIDI, it time-stamps events using a low-latency multimedia timer thread whose resolution is on the order of 2 [ms].[2] This functionality provides BoB with *raw solo input* data, a real-time stream comprised of the following pair of fields:

1. a discrete pitch value ($\texttt{note} - \texttt{on}$) or silence ($\texttt{note} - \texttt{off}$);
2. an *onset*, or equivalently, a *duration* time.

Since the agent sets tempo, time is immediately expressible in terms of the music's underlying beat; per-bar segmentation is trivial.

[2]On the *NT* operating system, a resolution within 2 to 4 [ms] can usually be maintained, although every hour or so, an occasional half-second glitch might occur.

3.1.2 Melodic Content

BoB's concern is handling melodic content in an intelligent manner. *Melodic content* is defined as the "musically appropriate" *solo transcription* that one would obtain if a reasonably competent improviser had been asked to write out their improvised solo using commonly employed musical notation. For example, the staves in Figure 1.1 were obtained by recording myself trading solos with myself and then transcribing what I heard.

For BoB to handle live performance, it must be able to clean up the raw solo input data, converting it into a format whose melodic content BoB can parse. Tasks that this cleaning process must perform are now described.

Monophonicity and Rests

First, the data stream must be monophonic. For example, on a MIDI keyboard, even when a monophonic melody is performed, the next $\texttt{note} - \texttt{on}$ might arrive a fraction of a second earlier than the last $\texttt{note} - \texttt{off}$. Another related issue is determining when a $\texttt{note} - \texttt{off}$ is long enough to be interpreted as a rest versus ignored. Using an approach similar to the one introduced by Longuet-Higgens (1987), I plan to handle both of these issues using simple threshold functions.

Rhythm Quantization

The serious challenge is to convert a monophonic stream of *performed durations* into their "musically sensible" temporal hierarchy. The problem is that rhythms are hierarchical, so events should align with various integer subdivisions of the music's beat. However, musicians rarely perform rhythms *mechanically*, which in the the non-interactive case, would refer to playing exactly what was written in a score. For example, a pair of eighth note durations might be "swung," the first duration held about twice as long as the second, even though altogether they occupy on the order of a quarter note worth of time. In contrast to playing from a score, with improvisation, one can expect rhythmic deviations to be all the more likely, for the musician is even less constrained.

3.1.3 Encodable Solo Input

A rhythmic quantizer performs a many-to-one mapping — transforming multiple performed duration sequences into the same hierarchical abstraction — so BoB only needs to be able to parse the limited set that the quantization module produces. Or, conversely, the set of *encodable* rhythms defined in Section 4.3.1 specifies those rhythms that BoB can parse, so the rhythmic quantizer will need to map inputs onto this range of outputs. Given the way encodable rhythms are defined, in 4/4 meter, quantization is a 1-in-32 classification task.

The more difficult part of the problem is to perform this mapping in a "musically sensible" manner. This task is difficult because rhythmic performance deviates from mechanical for a variety of reasons, be it expressive, aesthetics, or stylistic in nature, or the result of human error. As a result, performed duration deviations are not random (Desain, Aarts, Cemgil, Kappen, and van Thienen 1999), and so the traditional grid-based quantizer approaches used in many commercial sequencers can produce nonsensical results (Trilsbeek and Thienen 1999; Cemgil, Desain, and Kappen 2000).

The notation for encodable input segment n is:

$$s_{\mathsf{n}} = \left\langle s_{\mathsf{n},1}, \cdots, s_{\mathsf{n},\mathsf{t}}, \cdots, s_{\mathsf{n},\mathsf{T}''_{\mathsf{n}}} \right\rangle.$$

The n that subscripts T'' indicates that different segments can contain different numbers of events. The t-th event, $s_{\mathsf{n},\mathsf{t}}$, contains two fields: a discrete pitch or rest, and a duration value. When constructing the input features for bar r, the *most recently performed previous pitch* (MRP) is needed. Because of the way MRP is defined, the value for segment n, mrp_{n}, can be queried before the data for s_{n} arrives. Similar notation is used for both training data (bar n) and real-time interaction (bar r).

3.1.4 Simulated Live Input

For the time being, a musician's spontaneous input — **during both warmup and interactive solo trading** — is simulated by using data obtained from a *transcription file* that contains an encodable transcription of a particular musician's improvisation. For listening purposes, BoB provides an extra track of MIDI in addition to its own solo track so it can play back the musician's solo as well as its own.

With respect to timing and data flow, this simulation of a live musician behaves just as a live interaction would — a musician's bar of solo is only provided as input to the musically-intelligent components when it is played back. On the one hand, one might expect these simulations to produce behavior that looks better than it really should because the musician's calls and the training data are obtained from the same file. However, the "intimacy" that emerges from this "live" simulation will never be as good as with live musical exchange — for the musician does not "respond" to what BoB plays.

3.2 Off-line Learned Knowledge

Off-line Learned Knowledge (OLK) infers the parameters for a probabilistic mixture model from a set of warmup data collected from the musician prior to on-line interaction.

3.2.1 Data Collection

A typical warmup might last on the order of 5 to 10 minutes. Data collection involves the following steps:

1. The user specifies a restricted set of lead sheet information — e.g., tempo, rhythm, and (optional) harmonic information — that the computer will use to provide accompaniment.

2. The musician improvises freely on top of this accompaniment, the only restriction being that they synchronize to the accompanist's beat.

3. In real-time, BoB records what the musician plays, in total obtaining N encodable transcriptions:
$$S = \left\langle s_{1}, \cdots, s_{\mathsf{n}}, \cdots, s_{\mathsf{N}} \right\rangle.$$

4. Using the procedure described in Chapter 4, BoB converts S into training set
$$X = \left\langle x_{1}, \cdots, x_{\mathsf{n}}, \cdots, x_{\mathsf{N}} \right\rangle.$$

With simulated input, steps 1 through 3 are replaced by a parse of the appropriate transcription file.

3.2.2 Training Data

To justify the added complexity that musician-specific learning brings, BoB will be separately trained and evaluated on two musicians: Charlie Parker and Stephan Grappelli (Table 3.1). Parker, already studied in other machine learning contexts (Hörnel, Langnickel, Sandberger, and Sieling 1999; Rolland and Ganascia 1996), was a natural choice. Grappelli was chosen because I play the violin; he is one of my favorite improvisers on that instrument, and I would like to be able to "play with Grappelli through BoB" someday.

Table 3.1 also summarizes important properties of the transcription files compiled for both musicians. Each warmup was created from their improvisations over a single tune in order to increase the odds that the content would be relevant, i.e., that it would provide a cohesive set of behaviors from which the musician could be simulated "right now." For Parker, two independent improvisations over *Mohawk* were concatenated together[3] to form training set $X^{\boldsymbol{P}}$. For Grappelli, training set $X^{\boldsymbol{G}}$ was obtained from a single improvisation. A score of each source improvisation is presented in the appendix.

Musicians:		
	Charlie Parker	Stephane Grappelli
		
Instrument:	saxophone	violin
Genre:	Bebop	Swing
Transcription Files:		
Tune:	*Mohawk*	*I've Found a New Baby*
Harmonic Progression:	12-bar blues	32-bar AABA form
Key:	$B\flat$	F
Appendix:	E.1	F.1
Source:	*Mohawk I*; *Mohawk II* (Goldsen 1978)	*I've Found a New Baby I* (Glaser and Grappelli 1981)

Table 3.1: Musician Training Data

[3]Repeats were ignored.

3.2.3 Training Data and Harmonic Structure

In Section 2.4.3 I argued against presently giving BoB explicit harmonic knowledge. As a result, the current learning system knows nothing about the harmonic context in which a warmup session was collected. However, in both Parker's and Grappelli's improvisations, it is quite likely that the underlying harmonic context does influence how their underlying tonality unfolds in time — some bars' pitch content are more likely to be correlated with one another than others.[4]

In what follows, highlights of the harmonic structure found in *Mohawk* and *I've Found a New Baby* are presented. The point of this discussion is three-fold. My first intention is to shed light on the improvised contexts that take place in harmonic settings (as opposed to what Derek Bailey calls "free improvisation" (Bailey 1992)). Second, this structure is important because it is later used to musically assess the quality of learned tonal model (Section 7.2.1). Finally, at a high level, this structure can be used to modify a musician's call so that, when it is used to drive the system's search for a response, the response is more likely to fit into the underlying harmonic context.

Mohawk

Figure 3.1 illustrates how Charlie Parker played his improvisation in time. After an introductory *pick-up* note (not shown, bar 1), Parker sequentially played from bar 2 (top-most left box) to bar 121 (lower-most right box). The crucial point is that the solo is aligned in chunks, 12 bars per row. Each of these rows is another *chorus* of solo, and each chorus is structured in time with respect to the the tune's underlying *harmonic progression* (in the case of *Mohawk*, a 12-bar blues).

Parker's training data contains ten choruses. 1-5 were obtained from the first transcription and 6-10 were obtained from the second. During each transcription's initial chorus (1 and 6), the tune's main melody, i.e., its *head*, was played, following which Parker would improvise, each improvised solo differing significantly from the others. In contrast, aside from the fact that the second head was transcribed down an octave, choruses 1 and 6 were identically transcribed.

[4] This observation is part of what led me to consider the mixture model in the first place.

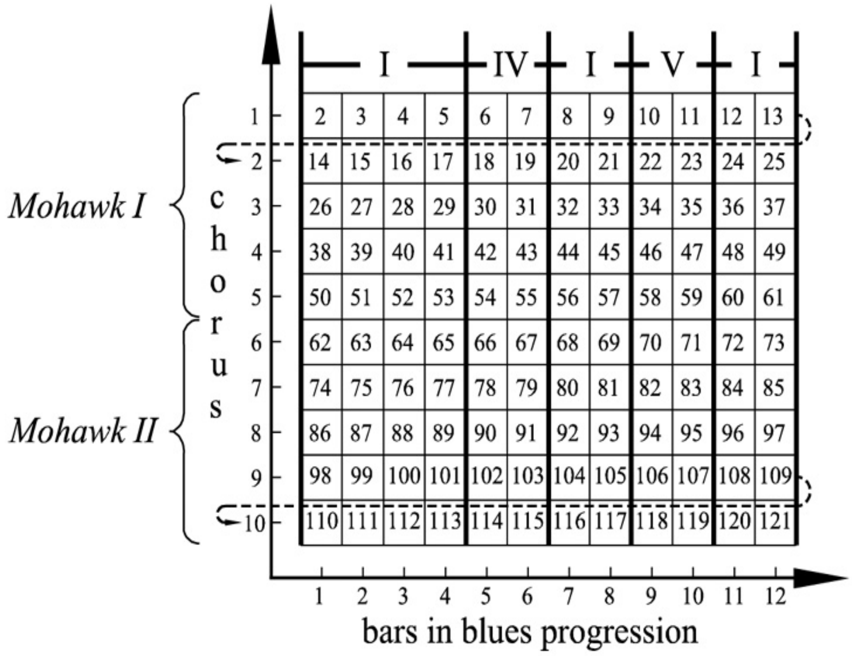


Figure 3.1: Parker's Training Data: Bars Structured in Time

The 12-Bar Blues

At a most basic level, the 12-bar blues progression can be divided into five key regions (Gridley 1988), each serving one of the following harmonic functions:

- *tonic*, I;
- *sub-dominant*, IV;
- *dominant*, V.

These regions, identified by the Roman numerals at the top of Figure 3.1, should be interpreted in the context of the tune's underlying key. For *Mohawk*, these functions correspond to the chords: B♭7, E♭7, and F7. These chords in turn contain the following pitch-classes: $\{B\flat, D, F, A\flat\}$, $\{E\flat, G, B\flat, D\flat\}$, and $\{F, A, C, E\flat\}$. Because these functions are aligned with the training data on a per-column basis, one expects harmonic context to influence how Parker's tonality unfolds in time.

Generally, the blues progression begins and ends on the tonic chord. The dominant and the sub-dominant chords, sharing only one common pitch class with the tonic, indicate regions of harmonic tension, setting up a context in which the tonic "cries out" to be returned to. Although each bar in column 12 is labeled as a tonic, in reality, resolution to the tonic only occurs when the improvisation finally comes to a close (chorus 5 and 10). The rest of the time, the accompaniment performs a *turn-back* — a special, short chord sequence that further sets the stage for returning to the tonic in bar 1 of the next chorus.

If this simple harmonic explanation was the end of the blues story, it might make sense to engineer this knowledge directly into BoB. To do so, however, would limit its generality somewhat (making the agent more of a blues expert). In reality, however, the series of chords transcribed for Parker's accompaniment changed much more quickly than these harmonic functions indicate.[5] As is the case with melodic improvisation, blues accompaniment has an infinite variety of realizations, chord substitutions, extensions, transformations, etc. While the simple harmonic view presented here is musically salient in the context of *Mohawk*, in general, to transform chords transcribed at the surface of a tune into this higher-level blues abstraction is difficult (Pachet 2000).

I've Found a New Baby

Grappelli's improvisation was played on top of another temporal structure commonly found in jazz: the 32-bar *AABA form*. Each chorus, comprises 32 bars, is broken into four equal parts, the third comprised of a harmonic progression that is significantly different than the other three, which are, roughly speaking, identical. Grappelli's training data contains four choruses. Although one can interpret his first chorus as the tune's head (as is typical in many of Grappelli transcriptions) this introductory chorus is a significantly embellished version of the tune's main melody.

3.2.4 Off-line Learning

Off-line, using the the procedure described in Chapter 5, BoB customizes itself to the user by fitting the parameters, Ω, of a mixture model to X. After learning, BoB uses this model to perceive and generate new solos on-line.

This approach assumes that the probability distribution over the user's behavior during warmup and solo trading does not change. However, because of the model's

[5]In Bebop, chords typically change twice per bar.

natural Bayesian interpretation, this assumption could easily be relaxed. A procedure for modifying Ω on-line, incorporating new bars of evidence into a parameter updating procedure as they arrive, is straightforward.[6]

3.3 On-line Interaction

During on-line interaction with the user, BoB's classification and generation techniques (Chapters 5 and 6) rely on using Ω to operationalize how it listens to what the musician plays and how it responds to what it hears "in kind."

3.3.1 Perception

The perception component converts each segment of a musician's solo into a more abstract representation. This process receives two inputs:

1. the parameters of the user model, Ω (from OLK);
2. an encodable sequence, s, and corresponding *mrp* (from the environment).

Two stages of output are produced:

1. a basic representation (chapter 4), which is comprised of a tree-based encoding, v, and related set of histograms, x.
2. a user-customized representation (chapter 5), which includes:
   (a) an estimate of which playing mode, y, is most likely to have generated x;
   (b) given this mode, the corresponding degree of *musical surprise* (Section 5.10.5), l, contained in x.

The left plot in Figure 3.2 shows how BoB might perceive a musician's fours. The discrete symbols emphasize each mode's discrete, nominal character. The right plot illustrates where most of the perception component's generalization takes place. In particular, many different transcriptions may map into the same user-specific playing mode (when a finer level of distinction is desired, musical surprise is available).

[6]What is more difficult is to on-line modify the number of components in the model.

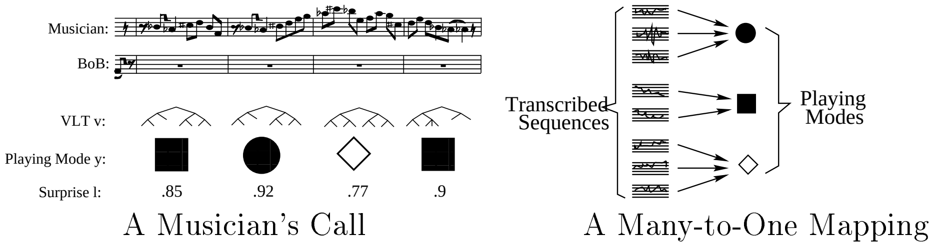


Figure 3.2: Perception

3.3.2 Local History

The history component is responsible for storing the current musical environment — the call — into a local, temporally ordered memory. For each bar r in a call, the following information is retained:

- a tree, v_{r}, and its related set of histograms, x_{r}, and mrp_{r};
- an estimated playing mode, y_{r}, and surprise, l_{r}.

With this information, the history component has the potential to compile more abstract sequences, e.g. $\langle y_{\mathsf{r}}, y_{\mathsf{r}+1}, \cdots \rangle$, $\langle mrp_{\mathsf{r}}, mrp_{\mathsf{r}+1}, \cdots \rangle$, etc.

3.3.3 Generation

The generator component creates a musically-appropriate, novel, and stimulating solo response to the musician's most recent call. BoB's generator produces a response a segment at a time.

Generation for bar r receives the following inputs:

- the very most recent context to link into, mrp_{r};
- a context that includes some perception, v, y, and l;
- the parameters of the user model, Ω;
- high-level search parameters (how long to search, etc.);
- a fairly general set of optional constraints (Section 6.3.5).

Generation produces two outputs:

1. transcription s_{r}, which is played back in real time during bar r;
2. the internal representation, v_{r}, y_{r}, l_{r}, and the new $mrp_{\mathsf{r}+1}$.

Abstract Cut-and-Paste

Figure 3.3 displays a very simple interaction scenario for how BoB might generate a response to a musician's call in the context of trading fours. In this scenario, *length*, the number of segments in both the call and the response, is four. Generation proceeds from left to right. In this figure, the second bar of response $(\mathsf{r}+5)$ has just been determined and is awaiting playback in the real-time buffer. The third bar of response $(\mathsf{r}+6)$ is currently being calculated.

This type of high-level interaction, referred to as *abstract cut-and-paste*, is perhaps naively simple because it assumes that an adequate response can be obtained by merely reusing a musician's call context in bar r as the input to the generator's response in bar $\mathsf{r} + length$. . However, there is nothing inherent in BoB's architecture that keeps it from accommodating more sophisticated interaction schemes (Section 6.4.3). Besides, one of the most exciting things about this scheme is that it offers a viable compromise to the conflict that arises when the user, who needs some degree of artistic control, and BoB, who needs an operational method for responding autonomously, interact. This compromise is only possible because the generator synthesizes two types of information: that which the musician directly controls via their call, and that which they indirectly affect through their warmup data.

Real-time Issues

With abstract cut-and-paste scenarios, each call/response pair must occupy an identical number of bars. In addition, each call must occupy at least two bars. This latter requirement arises because, during the musician's first call bar, BoB must wait to receive the entire segment before parsing and perceiving it. In subsequent call bars, multi-threaded techniques are used to simultaneously:

1. Buffer the currently performed call bar.
2. Perceive the previously played call bar.

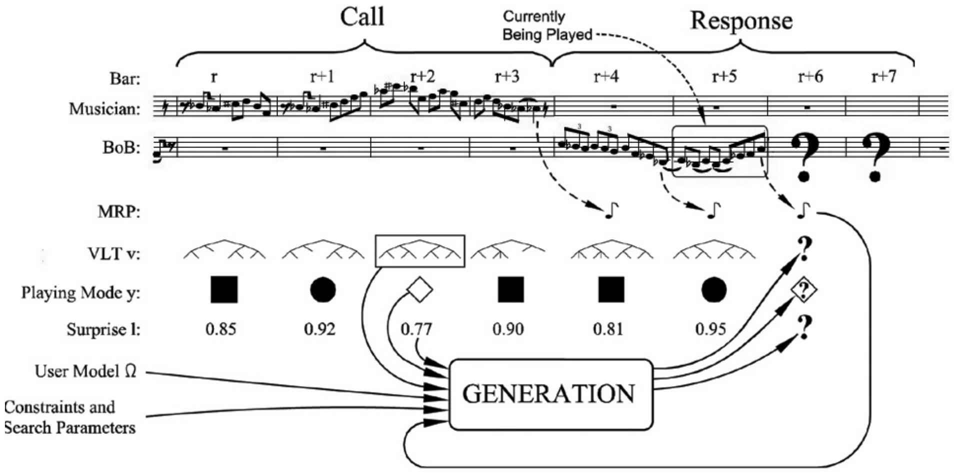


Figure 3.3: Generation: Call and Response

3. Use this perception to compute an appropriate response for that bar.

Harmonic Structure

When an external harmonic structure drives an interaction, the abstract cut-and-paste scheme described above is modified so as to consider this underlying context. Specifically, that part of the call's playing mode that relates to a segment's tonality is overridden by an appropriate abstraction from the training set.

For example, when BoB generates a response to a call that Parker played in the first four bars of the *Mohawk* progression, the tonal aspect that drives this response is taken from a "corresponding column" of training data (Figure 3.1). As before, melodic continuity and contour abstractions are copied directly from the musician's call. This modification illustrates one reason why a *conglomerative* modeling approach, i.e., an approach that individually considers tonal, intervallic, and contour-based aspects, is beneficial to the agent, offering an additional degree of power and flexibility. This modification is described further in Section 6.4.2.

Chapter 4

Basic Representation

This chapter introduces BoB's *basic representation*, those aspects of a solo's encoding that occur prior to learning. This chapter presents two data structures. The first is a tree that explicitly encodes a melodic segment's rhythmic hierarchy. The second is a conglomeration of histogram-based views that encodes deeper aspects of a melody's structure than are present when its melodic surface is presented as a string- or sequence-of-pitches. The primary purpose of these histograms is to provide an adequately rich set of features to the learning algorithm (Chapter 5), so that it can transform them into a more abstract, user-specific encoding that differentiates between salient aspects of their playing modes.

4.1 Overview

Figure 4.1 displays a segment of solo and its transformation into BoB's basic representation. This transformation is hierarchical in that each successive step produces a more abstract view of the melody than its predecessors:

1. obtain an encodable segment of solo, s, and transform it into *variable length tree* (VLT) v;

2. convert this tree into a *conglomerative* (CGL) *set of features*, x^{C}, which consists of:

   (a) a pitch-class (PC) histogram, x^{P};

   (b) an intervallic (INT) histogram, x^{I};

   (c) and a directional (DIR) histogram, x^{D}.

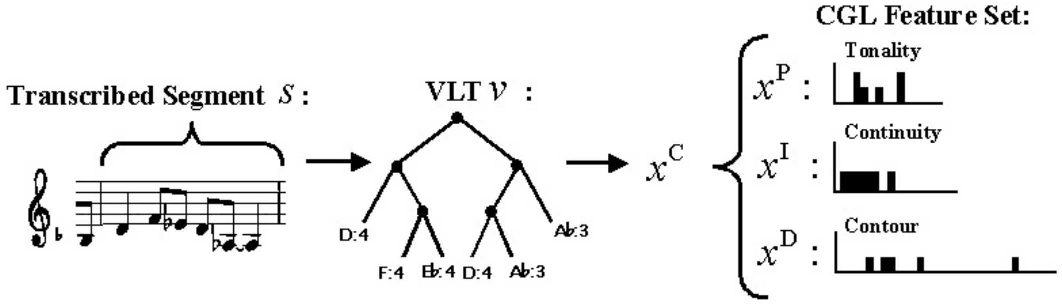


Figure 4.1: Converting a Bar of Solo Into Its Basic Representation

v is a hierarchical tree whose internal nodes encode a segment's rhythm and whose in-order walk of its leaves specifies the melody's pitch/rest sequence. By itself, this walk is nothing but a pitch/rest string, providing no generalization. x^{C}, however, maps this variable number of leaves into three fixed-size vectors so that deeper structural aspects of the melody might be operated upon. In particular,x^{P} provides a generalized tonal view of the sequence. x^{I} and x^{D} do the same for melodic continuity and contour.

In this thesis, segmentation is per-bar, but this basic representation can handle arbitrary segmentation schemes, an important issue in many temporal domains. As per-bar segments are relatively short, this scheme provides a highly-localized view of a solo. This representation was first introduced in (Thom 1999).

4.2 Segmentation Window Size

While smaller segmentation windows produce more local views, and thus can provide lower latency (which in turn enables the system to be more responsive), smaller windows necessarily contain less information because any averages inferred from them are based upon less data.[1] Clearly, window length can directly impact how a temporal model performs. In BoB's computational model, care is taken to treat window size as an arbitrary variable so that in future work the relationship between window size and performance can be easily investigated.

Computationally, arbitrary window sizes allow different solo segments to occupy

[1]A similar phenomenon occurs with Fourier Transforms, where the trade-off between time versus frequency resolution is a function of window length.

different time spans and/or circumscribe different numbers of events. This functionality is musically powerful because it allows more salient segmentation schemes to be considered (e.g., segment per-phrase or per-motive).

Because of BoB's real-time scheduler, which must know how long it has to think about each segment before responding, BoB's segmentation scheme is per bar. The disadvantage of this scheme is that, in jazz, motivic ideas often cross bar lines. However, since bars do correspond to predominant metrical patterns, this scheme has some musical merit. Per-bar segmentation has also been successfully used by other researchers, e.g., (Biles 1998; Franklin 2001; Hörnel, Langnickel, Sandberger, and Sieling 1999).

4.3 Variable Length Trees

A VLT provides the system's lowest-level view of a segment's melody. VLT v is obtained by parsing encodable input s with the `makeVLT` procedure (Appendix C). A VLT stores information using two different constructs:

1. **Leaf nodes** encode pitch information.
2. **Internal nodes** encode durational information.

A VLT is shown in Figure 4.1. Each internal node, $\bullet$, is connected to its children. Leaves are drawn at the bottom, and the root node is drawn at the top.

4.3.1 Internal Structure

A key aspect of VLTs is that their rhythms are entirely determined by the structure of their internal nodes. A specific node's duration is encoded by its location in a tree. In particular, at each split, the current node's duration becomes an appropriate subdivision of its parent's duration.

In detailing VLT rhythmic concepts, a 4/4 meter and per-bar segmentation scheme is assumed. Trees contain exactly four beats worth of solo, which simplifies various aspects of the tree's construction. Although it would be straightforward to extend these concepts to handle other meters and segmentation schemes, to do so will require some thought.[2]

Deciding how to limit rhythmic structure was based on my experience playing published improvised transcriptions (ranging from the blues, folk, jazz, bluegrass, swing, Cajun, and gypsy violinists covered by *Fiddler* magazine to artist-specific compendiums, e.g., Charlie Parker, Jeff Beck, Stephan Grappelli, Vassar Clements, Jimmy Page, Carlos Santana). A VLT's internal structure is restricted as follows:

- **Restriction I:** Each internal node must have 2 or 3 children.
- **Restriction II:** A leaf node can occupy no less than $\frac{1}{6}$ or $\frac{1}{8}$ of a beat.

These restrictions limit the number of duration nodes that can be used to construct a tree.

Those node durations that a VLT can represent are shown in Figure 4.2. These nodes are referred to as *valid* durations. In this figure, each duration is shown as both its fraction of a beat and its musical notation. Durations that descend to the left (long dashed lines) correspond to division by two, and those that descend to the right (short dashed lines) correspond to division by three. The top-most duration at tree-depth zero corresponds to a VLT's root node (occupying four beats). A tree's bottom-most leaf can never be smaller than either a thirty-second of a beat ($\frac{1}{8}$), or a sextuplet ($\frac{1}{6}$). When building and modifying trees, the values in this figure need to be accessible. Function:

$$\texttt{isValid}(num, den, dur)$$

is used for this purpose, only returning `true` when duration *dur* multiplied by *num* and divided by *den* is valid.

The trees that can be built using restrictions I and II define the set of transcribed rhythms that is *encodable*. Because individual nodes can be combined to form compound durations (Section 4.3.4), this set is rich enough to encode almost all of the transcriptions that I have encountered. A notable exception is Parker's and Grappelli's occasional use of the *pentuplet*.[3] To encode pentuplets, I modified their transcription, representing them as rhythms composed of two and three equal parts.

[2]For example, perhaps the rightmost tree in Figure 4.2 would be more appropriate in certain 3/4 meter contexts.

[3]The pentuplet divides a beat into five equal parts.

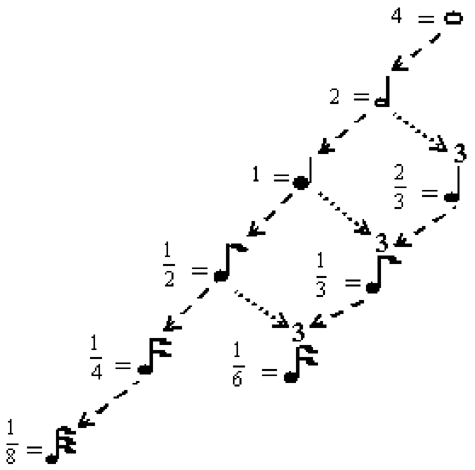


Figure 4.2: Node Durations (Per-bar; 4/4 Meter)

4.3.2 Uniqueness

Because there are multiple paths from the root to certain durations in Figure 4.2, an additional restriction on a VLT's internal structure is imposed:

- **Restriction III**: a VLT must be constructed from the smallest possible number of nodes.

This restriction ensures that a rhythm mapped into a VLT is *unique* or *canonical*.[4] For example, without this restriction, it would be ambiguous which tree should be used to encode the rhythm in Figure 4.3.

composed of 11 nodes composed of 12 nodes

Figure 4.3: A Rhythm With Two Realizations

[4]To prove that this additional definition provides a canonical encoding, one needs to show that for any encodable rhythm, one cannot produce two different trees using Restrictions I and II that, on the one hand, contain an identical number of nodes and, on the other hand, have unique internal structures.

4.3.3 Leaves

Leaf nodes store one of the following *leaf values*: a pitch, or a rest.

When a leaf stores pitch, it encodes it using the following fields:

- **pitch class**, a value from the set: $\{C, D\flat, D, E\flat, E, F, G\flat, G, A\flat, A, B\flat, B\}$. This scheme does not make pitch-spelling distinctions (e.g., F# is encoded as G♭).

- **octave**, an integer value in the range $[0, 7]$. This range is aligned so that the value 4 corresponds to the pitches between and including middle C up to the B above it.

In Figure 4.1, pitch class and octave fields are separated by a colon.

VLTs can only encode monophonic solos because each leaf is defined to contain at most one pitch value. With the exception of musical passages comprised of double-stops or chords, this encoding accommodates the transcriptions mentioned earlier. However, any additional expressive information that might be transcribed is lost. For example, dynamics and articulation — e.g., slurs, staccato, accents — are ignored.[5] Similarly ignored are pitch altering transcription devices, e.g., ghost, blue, or bent pitches are encoded as a discrete pitch alone.

4.3.4 Ties

Notice that there are two A♭:3 leaves in Figure 4.1. Both leaves are needed to represent the dotted-quarter note because this duration does not exist by itself in Figure 4.2. This example motivates why pitch values have an additional *tie* field.

When a pitched leaf is followed by another leaf whose pitch and octave are identical, a boolean *tie* variable indicates whether or not the two durations are treated as:

- **compound**, in which case the two leaves are played back as one longer tone;

- **composite**, in which case the two leaves are played back with a brief silence between them.

[5] A slur, which indicates that two distinct pitches should be played back smoothly, with no space separating them, should not be confused with the tie, which will be introduced to create compound durations.

With compound durations, the rhythms that a VLT can encode significantly increases.

When transcribing segment s into v, adjacent identical notes are assumed to be tied unless the first of the two is explicitly identified as composite. This default behavior provides an important semantic benefit when computing new solo segments in real time. In particular, when the last pitch in the most recently generated response bar is turned off solely depends on what BoB decides to do in the next response bar.

4.4 The Conglomerative Feature Set

The CGL feature set, x^{C}, converts a sequence that contains an arbitrary number of pitches into three fixed-size histogram vectors. The purpose of this abstraction is to transform the string-of-pitches in v into a set of local averages that captures deeper levels of structure than is available on the melody's immediate surface. This representation's primary function is capturing various aspects of the pitches' ordering in v. As a result, rhythm only indirectly affects x^{C} (influencing how many leaves there are and introducing syncopation); rests are ignored.

4.4.1 Pitch Sequences

Given an in-order walk, ρ_{r}, of the VLT in segment r:

$$\rho_{\mathsf{r}} = \left\langle \rho_{\mathsf{r},1}, \cdots, \rho_{\mathsf{r},\mathsf{t}}, \cdots, \rho_{\mathsf{r},\mathsf{T}'_{\mathsf{r}}} \right\rangle.$$

Pitch sequence ζ_{r} is defined to be the sequence that results when all the rest leaves are ignored:

$$\zeta_{\mathsf{r}} = \left\langle \zeta_{\mathsf{r},1}, \cdots, \zeta_{\mathsf{r},\mathsf{t}}, \cdots, \zeta_{\mathsf{r},\mathsf{T}_{\mathsf{r}}} \right\rangle.$$

When computing intervallic and directional histograms, the MRP will also be needed. For convenience, this value is referred to by index $\mathsf{t} = 0$, e.g., $\zeta_{\mathsf{r},\mathsf{t}=0}$.

4.4.2 Variable Sample Size

The number of counts distributed among a histogram's bins is its *sample size*,[6] sz:

$$sz = \sum_{\mathsf{m}} x_{\mathsf{m}}.$$

[6]Do not confuse a histogram's sample size, sz, with the number of histograms in a dataset, N.

Variable sample size indicates that different histograms may contain different numbers of counts.

4.4.3 Pitch-Class Histogram

The PC histogram records the distribution of pitch classes in ζ_{r}:

$$x^{\mathrm{P}}_{\mathsf{r,m}} = \#\Big\{1 \leq \mathsf{t} \leq \mathsf{T}_r | \mathsf{pc}(\zeta_{\mathsf{r,t}}) == \mathsf{m}\Big\}.$$

pc converts a pitch into an index that identifies its PC value (octave is ignored). $x^{\mathrm{P}}_{\mathsf{r}}$ captures essential aspects of the tonality in segment r. This abstraction is 0-order Markovian because it entirely ignores the ordering of pitches in the sequence. This histogram has $\mathsf{M}^{\mathrm{P}} = 12$ bins, one for each element in the pitch class set (Section 4.3.3). Sample size is determined by the number of events in segment r, $sz^{\mathrm{P}}_{\mathsf{r}} = \mathsf{T}_r$. A segment taken from Parker's training data and its corresponding VLT and PC histogram are shown in Figure 4.4.

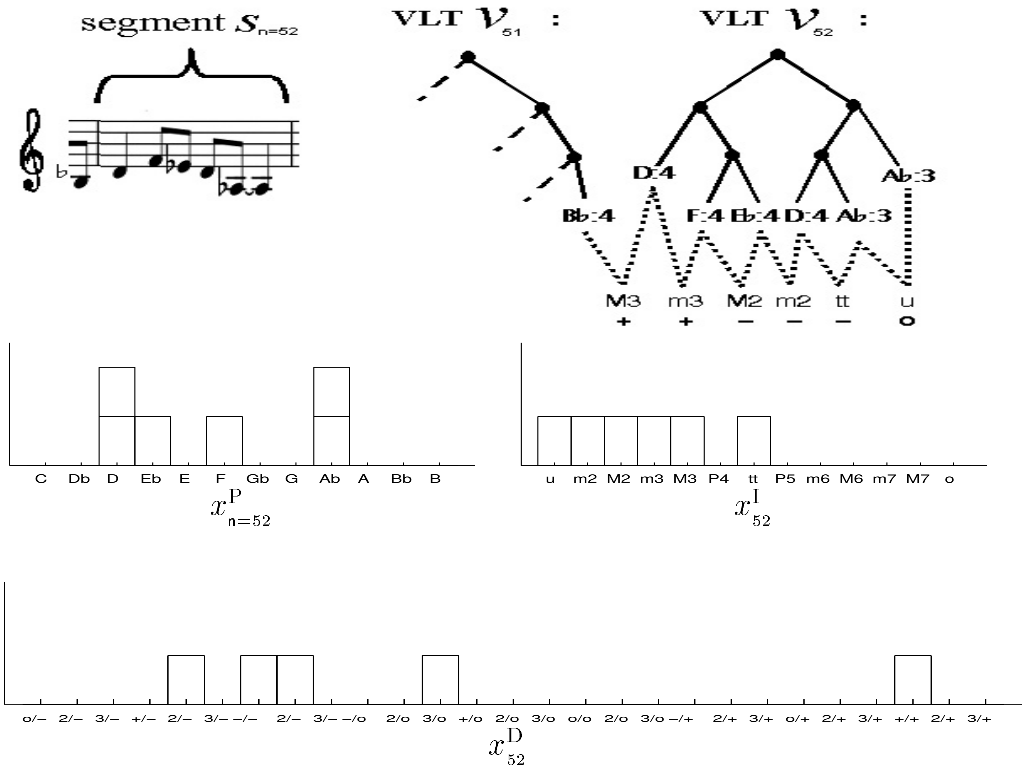


Figure 4.4: Conglomerative Feature Set Example

4.4.4 Interval Histogram

The INT histogram records the distribution of absolute intervals between adjacent pairs of pitches in ζ_{r}:

$$x^{\mathrm{I}}_{\mathsf{r,m}} = \#\left\{1 \leq \mathsf{t} \leq \mathsf{T}_r | \mathtt{abs}(\zeta_{\mathsf{r,t}}, \zeta_{\mathsf{r,t-1}}) == \mathsf{m}\right\}.$$

`abs` returns an index that identifies its absolute difference in semi-tones. $x^{\mathrm{I}}_{\mathsf{r}}$ captures essential aspects of the melodic continuity in segment r. This abstraction is 1st-order Markovian, concerned with pairs of adjacent pitches. Because $\zeta_{\mathsf{r,t=0}}$ is included in this calculation, $sz^{\mathrm{I}}_{\mathsf{r}} = sz^{\mathrm{P}}_{\mathsf{r}}$.

This histogram has $\mathsf{M}^{\mathrm{I}} = 13$ bins, one for each of the following intervals:

$$\{\mathrm{u}, \mathrm{m2}, \mathrm{M2}, \mathrm{m3}, \mathrm{M3}, \mathrm{P4}, \mathrm{tt}, \mathrm{P5}, \mathrm{m6}, \mathrm{M6}, \mathrm{m7}, \mathrm{M7}, \mathrm{o}\}.$$

All but 'u', 'tt', and 'o' are typical music notation. 'u', the unison interval, indicates that two adjacent pitches are identical. 'o' includes any internal greater than or equal to an octave. 'tt' refers to the tritone.

An INT example is also shown in Figure 4.4. The intervals for each pair of pitches are also identified by the jagged, dotted line that links the leaves in v. The intervals associated with this histogram's bins increase from left to right.

4.4.5 Melodic Direction

Each interval between a successive pair of pitches has a corresponding sign. Interval signs are recorded using an alphabet comprised of set:

$$\mathcal{D} = \{+, o, -\}.$$

Function $\mathtt{sym}(\zeta_{\mathsf{t-1}}, \zeta_{\mathsf{t}})$ assigns symbols to intervals as follows:

- '+' indicates intervals with *upwards* motion ($\zeta_{\mathsf{t-1}} < \zeta_{\mathsf{t}}$);
- 'o' indicates the unison interval, i.e., *zero* motion ($\zeta_{\mathsf{t-1}} == \zeta_{\mathsf{t}}$);
- '−' indicates intervals with *downwards* motion ($\zeta_{\mathsf{t-1}} > \zeta_{\mathsf{t}}$).

`sym` is used to create *direction sequence*:

$$d_{\mathsf{r}} = \left\langle d_{\mathsf{r},1}, \cdots, d_{\mathsf{r,t}}, \cdots, d_{\mathsf{r,T_r}} \right\rangle \tag{4.1}$$

from pitch sequence ζ_{r}.

4.4.6 Melodic Direction Histogram

The DIR histogram records essential aspects of the melodic contour in segment r. In particular, $x_{\mathsf{r}}^{\mathrm{D}}$ records the transitions that d_{r} produces when parsed by the Markov Chain of Figure 4.5. This abstraction provides a variable sized Markovian view of sequence ζ_{r}, considering anywhere from two to four of the most recent pitch values depending on context. x^{D} has $\mathsf{M}^{\mathrm{D}} = 27$ bins, one for each transition in Figure 4.5.

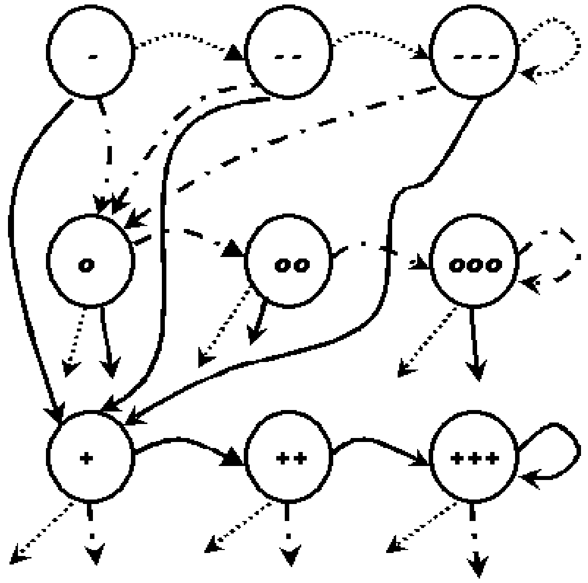


Figure 4.5: Relevant Directional History Markov Chain

The *relevant directional histories* that this chain observes come from the set:

$$\mathcal{H} = \{+, ++, +++, o, oo, ooo, -, --, ---\}.$$

In total, $n_{\mathcal{H}} = 9$ states. This Markov chain is highly constrained. Rather than being fully connected, there are only three transitions out of each state, one for each symbol in $\mathcal{D}$. (In contrast, a fully connected chain would have 81 transitions.) These constraints reduce the dimension of x^{D}, which makes learning its structure more feasible and sampling from its distribution more efficient.

The dotted arcs in this figure correspond to downwards motion transitions (solid arcs to upwards motion; dot-dashed arcs to zero motion). It would clutter up the graph to connect every transition to its appropriate input state. Rather, all dotted arcs that do not terminate end in state '$-$'. (Solid arcs end in state '$+$', and dot-dashed arcs end in state 'o'.) From this figure, it is easy to see what aspects of melodic contour DIR bins capture — they count repeated directions of up to length three, resetting back to length one anytime a change in direction occurs.

Each bin in DIR records a different tuple, $\langle i, j \rangle$, where $i \in \mathcal{H}$ and $j \in \mathcal{D}$. A one-to-one mapping between index $\mathtt{m}$ and this tuple is obtained via $\mathtt{m} = \mathtt{dInd}(i, j)$. Function $\mathtt{hs}$ is used to recursively generate the sequence of states, h_{r}, that are obtained when d_{r} is parsed by the Markov chain:

$$h_{\mathsf{r}} = \left\langle h_{\mathsf{r},1}, \cdots, h_{\mathsf{r},\mathsf{t}}, \cdots, h_{\mathsf{r},\mathsf{T}_{\mathsf{r}}-1} \right\rangle.$$

With this sequence in hand, x^{D} is calculated:

$$x^{\mathrm{D}}_{\mathsf{r},\mathsf{m}=\mathtt{dInd}(i,j)} = \#\left\{2 \leq \mathsf{t} \leq \mathsf{T}_r | h_{\mathsf{r},\mathsf{t}-1} == i \text{ and } \mathtt{d}_{\mathsf{r},\mathsf{t}} == \mathtt{j}\right\}.$$

This equation's indices range from 2 to T_r; $sz^{\mathrm{D}}_{\mathsf{r}} = sz^{\mathrm{P}}_{\mathsf{r}} - 1$.

```
hs(h_{t-1}, d_t)
    if t == 0
        len = 1
        sym = d_t
    else
        len = number of repeated symbols in state h_{t-1}
        sym = direction symbol associated with state h_{t-1}
        if sym == d_t
            len = min(len + 1, 3)
        else
            len = 1
            sym = d_t
        end
    end
    hist = that h ∈ H that contains sym repeated len times in a row
    return hist
```

Table 4.1: A Recursive Relevant History Procedure

A DIR example is also shown in Figure 4.4. Bin labels in this figure are abbreviated. The '/' in each label separates a relevant history (left of slash) from its melodic direction transition (right of slash). For example, bin '+/−' is the downwards transition out of state '+'. The next two bins to the right, '2/−' and '3/−', correspond to longer upwards relevant histories, i.e., '++' and '+++' respectively. The transitions recorded by these bins are also downwards.

4.4.7 Conglomerative Histogram

The CGL tuple comprises the three preceeding histograms. x^{C} is obtained for segment r using function `featuresCGL` (Table 4.2).

$\boxed{\texttt{featuresCGL}\left(mrp_{\mathsf{r}}, \zeta_{\mathsf{r}}\right)}$
calculate x^{P} from ζ_{r}
$\zeta' = \zeta_{\mathsf{r}}$, with mrp_{r} inserted onto the front
calculate x^{I} from ζ'
calculate x^{D} from ζ'
$x^{\mathrm{C}} = \langle x^{\mathrm{P}}, x^{\mathrm{I}}, x^{\mathrm{D}} \rangle$
$\texttt{return}\ x^{\mathrm{C}}$

Table 4.2: The `featuresCGL` Pitch Sequence to CGL Algorithm

In total, x^{C} has:

$$\mathsf{M}^{\mathrm{C}} = 52 = \mathsf{M}^{\mathrm{P}} + \mathsf{M}^{\mathrm{I}} + \mathsf{M}^{\mathrm{D}} = 52$$

bins and contains:

$$sz_{\mathsf{r}}^{\mathrm{C}} = sz_{\mathsf{r}}^{\mathrm{P}} + sz_{\mathsf{r}}^{\mathrm{I}} + sz_{\mathsf{r}}^{\mathrm{D}} = 3 \cdot \mathsf{T}_{\mathsf{r}} - 1$$

counts.

4.4.8 Syncopation and Repeated Tones

Syncopated rhythms must be encoded as compound rhythms, and so the pitch class histograms built from these rhythms necessarily contain more counts than their non-syncopated counterparts would (e.g., rather than having one A♭:3 count in Figure 4.1, there are two). The increased PC weighting for syncopation makes sense because the syncopation tends to draw more attention to these pitches. Compound rhythms also produce unison interval counts and relevant histories that contain zero motions.

Currently, no distinction between compound and composite durations are made: both produce additional PC counts, unison intervals, and zero motion trends. However, *note repetition*, e.g., composite durations, plays an important part in music perception (Cross 1999; Narmour 1990). At some point, syncopation and repetition induced counts will need to be distinguished.

4.5 The Complete Basic Representation Algorithm

BoB's real-time basic perception algorithm, `basicRepr`, is outlined in Table 4.3. Initially, tree v is empty. `makeVLT` constructs this tree from s_{r}. From the resulting tree, leaf and pitch sequences are constructed. x^{C} is constructed from this pitch sequence and the MRP. This table provides insight into the reasoning behind the various primed sequence lengths. Assuming that s_{r} is in its smallest possible form: $T''_r \leq T'_r \leq T_r$.

$\boxed{\texttt{basicRepr}\Big(mrp_{\mathrm{r}}, s_{\mathrm{r}} = \langle s_{\mathrm{r},1}, \cdots, s_{\mathrm{r},\mathrm{t}}, \cdots, s_{\mathrm{r},\mathrm{T}''_{\mathrm{r}}} \rangle\Big)}$

- v is empty
- remove the first element from s_{r} and store it in *curr*
- make d the total duration of a single bar
- $\texttt{makeVLT}(curr, s_{\mathrm{r}}, v, d)$
- obtain $\rho_{\mathrm{r}} = \langle \rho_{\mathrm{r},1}, \cdots, \rho_{\mathrm{r},\mathrm{t}}, \cdots, \rho_{\mathrm{r},\mathrm{T}'_{\mathrm{r}}} \rangle$ from v
- obtain $\zeta_{\mathrm{r}} = \langle \zeta_{\mathrm{r},1}, \cdots, \zeta_{\mathrm{r},\mathrm{t}}, \cdots, \zeta_{\mathrm{r},\mathrm{T}_{\mathrm{r}}} \rangle$ from ρ_{r}
- $x^{\mathrm{C}} = \texttt{featuresCGL}(mrp_{\mathrm{r}}, \zeta_{\mathrm{r}})$
- $\texttt{return}\ \langle v, x^{\mathrm{C}} \rangle$

Table 4.3: The Basic Representation Algorithm

4.6 Discussion

4.6.1 Transcribed Melodic Segments and the VLT

A VLT's highly restricted internal structure provides important real-time benefits, reducing the computation associated with:

- transcription, because fewer encodable rhythms mean that fewer transcpriptions need to be considered;
- tree building, because a tree's maximum size is bounded;
- generation, because the search space over possible solos is significantly restricted.

The cost paid for these benefits is that BoB can only function when it receives encodable transcriptions as inputs. Another downside is that expressive aspects of a performance, e.g., accent, articulation, dynamics, timbre — important parts of the improvised music making experience — are ignored. This lack of expressive power is a primary reason why BoB's current audio demos sound so mechanical.[7] While these are significant drawbacks, I believe there are compelling reasons to investigate computer improvisation on top of such a restricted bases. Initially, it was my own improvisational experiences that motivated this line of thinking. For example, when I perform a transcription of Parker or Grappelli, my realization of it is much more expressive than what is printed on the page. Somehow, other aspects of my musical being are able to combine with this abstraction to produce a powerful result. Similarly, when I look at these transcriptions, using them as a guide when improvising, their content has a strong impact on the types of notes, rhythms, and motivs that I play, even though I am spontaneously making up my solo. In short, the transcription abstraction is a very powerful one, even in improvised contexts.

Another motivation for this agenda is the observation that in music research, performance has only recently started to receive much attention, and people are finally beginning to recognize the value in doing so. Performance investigations often compare performed timings to scores (Heijink, Desain, Honing, and Windsor 2000; Widmer 2000). Although an effective argument can be made that an obsessive preoccupation with written scores — for example, to the extent that audio content is ingored — might hamper computer music research, it is the presence of scored notation that allows expressive deviations to be quantified in the first place. While in improvised music it is more difficult to justify why a spontaneous melody should be viewed in a scored notation, I believe that developing this abstraction is a crucial next step to extending what is rigorously researched in music.

Another important benefit of VLTs is that they present rhythmic structure in an explicit, hierarchical manner. In contrast, many computer systems require that each musical pitch occupy a fixed duration (Franklin 2001; Hörnel, Langnickel, Sandberger, and Sieling 1999), and if a system is to generate a long note, it must know how to generate a long sequence of identical, tied pitches. BoB's explicit hierarchy provides a much more succinct rhythmic view.

With this encoding, it is also much more natural to think about rhythmic modification — merely grow or collapse some part of a tree's fringe (Section 6.1). Marsden

[7]While people have commented that much better playback could have been easily achieved — swinging eighth-notes, improving the accompaniment, using better MIDI patches (or, better yet, having a real musician perform BoB's solos) — I have resisted the temptation to make these modifications because it is in this state that listeners hear the same abstraction that the computer does.

(1999) pointed out the value of thinking about melodies in terms of these types of embellishments, but has yet to build an algorithm that can automatically infer how such embellishments could be used to relate various segments of music (he could easily generate examples by hand). The VLT framework provides one mechanism for exploring the operationalization of such ideas.

4.6.2 The Conglomerative Feature Set

CGL transforms a VLT's string-of-pitches into a set of averages that provides a more general view of local tonal, intervallic, and contour-based behavior. Initially, the appropriateness of this scheme was motivated by my sense that, when improvising with others or when transcribing improvised solos from CDs, I often remembered a more localized, general essence of the solo's content than an exact memory of individual notes themselves.

Others have recognized the importance of providing multiple views of a pitch sequence. For example, Cambouropoulos, Crawford, and Iliopoulos (1999) examined the pros and cons of various pitch-, intervallic-, and contour-based encodings. Sophisticated encodings that combine pitch-class, intervallic, and directional based views have also been used in real systems, e.g., (Hörnel and Menzel 1998). Since each view is readily perceived by humans (Deutsch 1999; Krumhansl 1990; Bartlett 1993), to ignore any one of them might violate a user's musical common sense, making it more difficult for them to "suspend their disbelief." In addition, by explicitly including each of these multiple, musically meaningful viewpoints into the same encoding scheme, the system becomes easier to understand and extend.

The histogram generalization mechanism ignores a segment's exact, pitch-by-pitch temporal behavior, focusing instead on more locally general ideas about the segment's scalar, intervallic, and directional properties. Thus, it is not surprising that different notions of similarity emerge when using string-of-pitches and CGL-based features. String-of-pitches based comparisons naturally lead to an *edit-based* distance metric, e.g., count how many insertions and deletions are needed to transform one sequence into another (Rolland and Ganascia 1996; Rolland and Ganascia 2000; Mongeau and Sankoff 1990). Although to some extent CGL-based comparisons can be interpreted in an edit-based framework, for insertions and deletions add, remove and/or rearrange histogram counts, an important distinction is that it is straightforward to probabilistically model histograms. As demonstrated in Chapter 7, when real music data is modeled in this way, a coarser, less "brittle" view of melodic likeness emerges.

The fact that histograms transform variable-sized sequences into fixed-dimensional feature vectors also impacts the types of machine learning algorithms that can be used. Typically, variable-sized inputs require some kind of decomposable, recursive learning technique, e.g., (Elman 1991), and such technologies are notoriously hard to train. Many in the computer music community have avoided this issue by restricting themselves to fixed-size inputs, either continuously adjusting the window size to automatically contain a fixed number of events, e.g., (Rowe 1993), or limiting oneself to a specific duration and resolution of inputs, e.g., (Franklin 2001; Hörnel, Langnickel, Sandberger, and Sieling 1999). The VLT/CGL combination provides a more flexible and powerful way to learn structure from a highly structured, temporal, contextual medium.

On the other hand, this fixed-size transformation is, by definition, ignorant of certain aspects of a segment's temporal ordering — this is the very reason it generalizes. As a result, it can be difficult or impossible to construct a viable pitch sequence from a CGL feature vector (Section 6.3.2). Together, fortunately, the trio of histograms provides a fair amount of temporal information (ranging from 0- to 3rd-order Markovian), and this is why BoB can generate pitch sequences that sound as good as they do.

Chapter 5

Learning Musician-Specific Perception

This chapter introduces the mechanism by which BoB transforms its basic representation of a solo segment into a user-specific representation. A probabilistic *mixture model* composed of *Multinomial* probability distribution components is introduced for clustering histograms that contain an arbitrary, variable numbers of counts. An Expectation-Maximization (EM) based method is derived to infer the parameters of this model, enabling a user-specific clustering function to be learned from the user's warmup data.

From a ML point of view, this model's primary distinction is that it treats all counts within a given histogram as correlated. Because of this distinction, the model provides more information to the learner than a method like *Autoclass* would because all counts are explicitly modeled as coming from the same underlying distribution. This fact is especially important because of the inherent sparsity of the training data that BoB is expected to collect. Such sparsity-related issues are investigated by first introducing simulation-based methods for quantifying a learning solution's quality, and then using these methods to quantify learning performance in the context of the transcriptions of Charlie Parker and Stephane Grappelli.

This model is important to computational musical intelligence because it provides a mechanism for clustering based upon the content of an entire segment. This technology is used to customize BoB's solo-trading model to its user, partitioning the space of conglomerative histograms into distinct playing modes that capture important distinctions in tonal, intervallic, and directional usage.

5.1 Overview

Figure 5.1 displays a many-to-one mapping of conglomerative features into discrete clusters. This mapping is customized to a particular user by fitting a mixture of Multinomial components to the PC, INT, and DIR histograms collected for them during a warmup session. Since these histograms' class memberships are unknown, learning is *unsupervised* — estimation amounts to partitioning their tonal, intervallic, and directional trends so that the training data is "well explained." This clustering scheme was first presented in detail in (Thom 2000b) and (Thom 2000a).

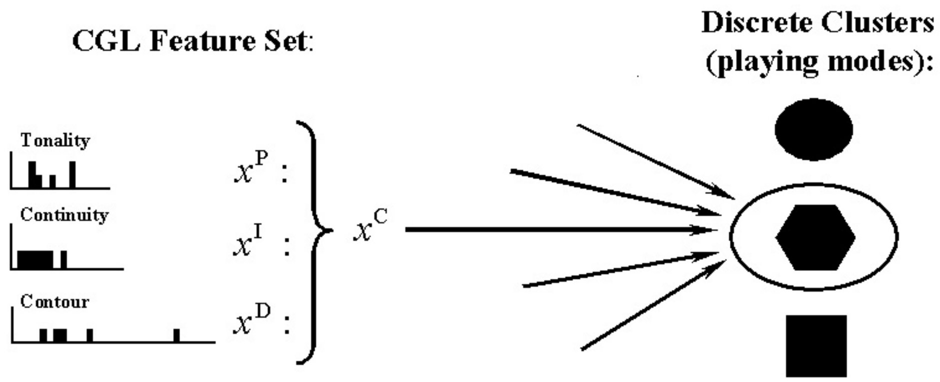


Figure 5.1: User-Specific Playing Mode Classification

The purpose of this mapping is to capture the various playing modes that the musician uses during different parts of their improvisations to make their solos interesting. For example, a user's playing mode might amount to their preference for certain tones in a musical scale, arranged in such a way that a particular melodic shape and set of intervals is used. The terms *cluster, class, component*, and *playing mode* are used interchangeably.

5.2 Training Data and Sparsity

BoB is independently trained and evaluated on Parker and Grappelli using the datasets described in Section 3.2.2, $X^{\boldsymbol{P}}$ and $X^{\boldsymbol{G}}$ respectively. Both datasets are relatively sparse in terms of their dimensionality (the number of PC, INT, and DIR bins). *Training set sparsity* concerns how many histograms that the dataset contains, and *sample size sparsity* concerns how many counts are distributed among each histogram's bins. Sparsity has an important impact on learning performance, so each of these aspects is further quantified to formalize what kind of "representative data" BoB should be able to learn from.

5.2.1 Training Set Sparsity

Those bars that produce VLTs with less than two pitch leaves are not included in the training data because their DIR histograms contain no counts. A transcription's initial pick-up bar was also not used. The remaining bars are used to construct PC, INT, and DIR datasets, producing a total of $\mathsf{N}^{\boldsymbol{P}} = 119$ histograms for Parker, and $\mathsf{N}^{\boldsymbol{G}} = 126$ for Grappelli.

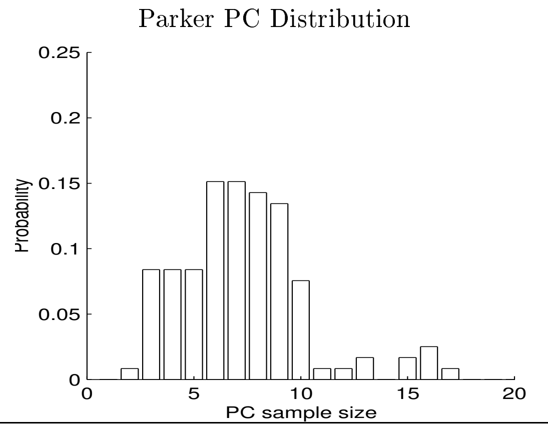
5.2.2 Sample Size Sparsity

The distribution over sample size for PC histograms are summarized in Table 5.1 for both Parker and Grappelli.[1]. To get a feel for how sparse these histograms are, see Figure 4.4.

Sample size sparsity is caused by BoB's per-bar segmentation window, which involves the adaptation versus averaging trade-off discussed in Section 4.2. In the context of histogram clustering, this trade-off becomes even more important because the degree to which histograms can be segregated into distinct clusters will depend on their underlying segments coinciding with unique and distinctive trends. Although larger windows provide more data upon which to estimate a histogram's underlying generative component, they increase the odds that several competing trends will contribute to the same histogram's counts, muddling up the very structure that is to be inferred.

[1] The PC distribution completely specifies how the INT and DIR distributions behave.

		PC sz Statistics							
		Parker				Grappelli			
	dim	Ave	Std	Min	Max	Ave	Std	Min	Max
PC	12	7.3	3.0	2	17	6.9	1.8	2	12
INT	13	7.3	3.0	2	17	6.9	1.8	2	12
DIR	27	6.3	3.0	1	16	5.9	1.8	1	11



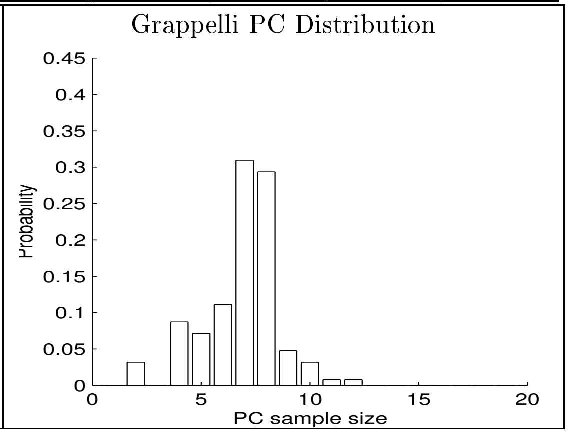


Table 5.1: Representative Sample Sizes

5.3 Clustering Background

In the last thirty years, the quantity of literature on cluster analysis has exploded, with contributions coming from a wide range of different fields e.g., (Dubes and Jain 1980; Kaufman and Rousseeuw 1990; Duda and Hart 1973). McLachlan and Basford (1988) presents a good overview of the mixture model based clustering approach.

A definition of cluster analysis is as varied as the field's interdisciplinary nature. I employ (McLachlan and Basford 1988)'s definition:

> In cluster analysis, multivariate techniques are used to create groups amongst the entities, where there is no prior information regarding the underlying group structure, or at least where there is no available data from each of the groups if their existence is known. (page 6)

In machine learning terms, this means that no data-points are labeled. The algorithm that operates upon the data infers these labels, an unsupervised learning scenario.

Clustering algorithms can be categorized as either hierarchical (tree based) or non-hierarchical (fixed partition). In the tree-based scheme, an algorithm attempts to grow some sort of tree to "explain" the data, each level in the tree merging or splitting a particular cluster into a sub- or super-set. From root to leaves, this tree provides a monotonically increasing strength of clustering as more and more clusters are formed. This type of strategy will always optimize a route between the root and leaf extremes (McLachlan and Basford 1988) (page 7).

In the fixed partition scheme, the number of clusters, K, is fixed up front. Learning amounts to searching for an optimal clustering of the data into this many partitions. The sole optimization focus is finding structure for K individual groups in the data:

> concentrat[ing] on the role of clustering where the partitioning of the data into relatively homogeneous groups is a data reduction exercise in its own right, or part of an attempt to shed light on the phenomena of interest through a particular grouping. (McLachlan and Basford 1988) (page 6)

Regardless of which scheme is used, the open question of how to "best" pick the number of clusters in dataset X remains. However, once K is chosen, either clustering strategy provides a mechanism for mapping dataset X into labeling Y.

5.4 Clustering With Mixture Models

Mixture models provide a fixed-partition based approach to clustering. This approach is *model-based* in that a different component probability distribution is estimated for each cluster. In addition to the standard probabilistic benefits, e.g., knowledge about uncertainty, mixture models are also *generative*, meaning that new data can be sampled from the model.

Mixture model G is composed out of some fixed number of underlying component distributions K:

$$\mathrm{G} = \left\langle \mathrm{g}_1, \cdots, \mathrm{g}_{\mathsf{k}}, \cdots, \mathrm{g}_{\mathsf{K}} \right\rangle.$$

A specific model is defined by parameter Ω, composed out of:

- **mixture weights**: $\pi = \left\langle \pi_1, \cdots, \pi_{\mathsf{k}}, \cdots, \pi_{\mathsf{K}} \right\rangle$. π_{k} specifies how likely component g_{k} is to be chosen as a data-point's generator;
- **component parameters**: $\theta_1, \cdots, \theta_{\mathsf{k}}, \cdots, \theta_{\mathsf{K}}$. θ_{k} specifies the parameterization for the g_{k}-th component.

5.4.1 Sampling

Mixture models assume that data is generated according to a two stage sampling process. First, a component is chosen:

$$y \sim \mathrm{Mnom}(1, \pi). \tag{5.1}$$

Next, this component is used to produce feature vector x:

$$x \sim \mathrm{g}_{\mathsf{k}=\mathsf{y}}(\theta_{\mathsf{k}}). \tag{5.2}$$

The Mnom function is fully described later (Section 5.7.1). For now, suffice it to say that equation 5.1 selects from one of the following outcomes:

$$y \in \{1, 2, \cdots, \mathsf{k}, \cdots, \mathsf{K}\}.$$

It will sometimes be convenient to present y in its equivalent *indicator vector* form:

$$z = \Big\langle z_1, \cdots, z_{\mathsf{k}}, \cdots, z_{\mathsf{K}} \Big\rangle,$$

where z is defined so that all dimensions have value zero except for $z_{\mathsf{k}=y} = 1$. Notation y and z will be used interchangeably. Thus, whenever y appears, indicator vector z is implied and vice versa.

Under this sampling scheme, the mixture model's joint probability is:

$$\Pr(x; y|\Omega) = \pi_{\mathsf{k}=y} \cdot \Pr\Big(x|\mathrm{g}_{\mathsf{k}}(\theta_{\mathsf{k}})\Big).$$

Marginalizing produces the unlabeled likelihood:

$$\mathrm{u}(x|\Omega) = \sum_{\mathsf{k}} \Pr(x; y = \mathsf{k}|\Omega). \tag{5.3}$$

When a labeled estimate is available, one can also specify the labeled likelihood:

$$\mathrm{l}(x|y; \Omega) = \Pr(x|y = \mathsf{k}; \Omega) = \Pr\Big(x|\mathrm{g}_{\mathsf{k}=\mathsf{y}}(\theta_{\mathsf{k}})\Big). \tag{5.4}$$

5.4.2 Clustering and Posteriors

Since the joint defines a probability distribution over $\langle x, y\rangle$, it provides the knowledge needed to cluster a dataset. In particular, for a given x, Bayes rule is applied to produce posterior distribution vector:

$$\psi = \Big\langle \psi_1, \cdots, \psi_{\mathsf{k}}, \cdots, \psi_{\mathsf{K}} \Big\rangle,$$

where:

$$\psi_{\mathsf{k}} = \Pr(y = \mathsf{k}|x;\Omega) = \frac{\Pr(x; y = \mathsf{k}|\Omega)}{\mathrm{u}(x|\Omega).} \tag{5.5}$$

The posterior is used to obtain cluster estimate y:

$$y = \mathtt{Argmax}_{1\leq \mathsf{k} \leq \mathsf{K}}(\psi_{\mathsf{k}}). \tag{5.6}$$

Since y is an estimate, it does not necessarily correspond to the component that actually generated x.

When, however, data is simulated, as will be the case when a particular learned parameter's quality is assessed (Section 5.7.5), both the data's generative components and the model's parameters are known. To identify this situation, variables are underlined (indicating that they are based upon the ground truth). Thus, with $\langle x, \underline{y} \rangle$ it is known that $\mathrm{g}_{\mathsf{k}=\underline{\mathsf{y}}}(\underline{\theta}_{\mathsf{k}})$ generated x. Similarly, $\underline{\Omega}$ indicates that x was sampled from $\mathrm{G}(\underline{\Omega})$.

When equation 5.6 is based on $\underline{\Omega}$, it is an *optimal Bayes classifier* — the number of misclassifications it makes will, on average, be minimal (DeGroot 1970). An optimal classification is additionally identified by a hat, $\underline{\hat{y}}$ (indicating that it is an estimate based upon ground truth).

5.4.3 Learning a Mixture Model from Training Data

When learning a mixture model from training set $X^{\boldsymbol{Tr}}$, the following are hidden:

- clustering $Y^{\boldsymbol{Tr}}$;
- parameters Ω;
- various structural assumptions, e.g., do all g_{K} come from the same family, and if so, from which family? what is K? etc.

Roughly speaking, once some structural assumptions are settled upon, learning boils down to estimating each θ_{k} and π_{k} using those data-points that are most likely to have been generated by the same component, g_{k}.

As a result, when the underlying components really are quite different from one another, the data is easier to separate and the model should be easier to learn. Ultimately, learning will be cast as an optimization problem, adjusting Ω to make a dataset likelihood function as large — as *fit* — as possible.

It is assumed that all data-points are independently sampled from $\mathrm{G}(\Omega)$, so labeled and unlabeled dataset likelihoods are multiplicative:

$$\begin{aligned} \mathrm{U}(X|\Omega) &= \prod_{\mathsf{n}} \mathrm{u}(x_{\mathsf{n}}|\Omega) \\ \mathrm{L}(X|Y;\Omega) &= \prod_{\mathsf{n}} \mathrm{l}(x_{\mathsf{n}}|y;\Omega) \end{aligned} \tag{5.7}$$

By supporting an additional *"fake" labeled dataset,* $\langle X, Y\rangle^{\boldsymbol{Fk}}$, the learning algorithm can automatically incorporate prior knowledge, which allows one to easily introduce bias. In total, the fitness function to optimize is:

$$\mathrm{F}(X^{\boldsymbol{Tr}}; X^{\boldsymbol{Fk}}|Y^{\boldsymbol{Fk}};\Omega) = \mathrm{U}(X^{\boldsymbol{Tr}}|\Omega)\cdot \mathrm{L}(X^{\boldsymbol{Fk}}|Y^{\boldsymbol{Fk}};\Omega).$$

These variable names were chosen to be easy to remember, i.e., *U*nlabeled and *L*abeled likelihoods, and the training-plus-fake-dataset *F*itness function.[2]

Learning Ω amounts to optimizing:

$$\mathtt{Argmax}_{\Omega}\Big(\mathrm{F}(X^{\boldsymbol{Tr}}; X^{\boldsymbol{Fk}}|Y^{\boldsymbol{Fk}};\Omega)\Big). \tag{5.8}$$

This estimate is only guaranteed to locally approach optimal because, with unlabeled likelihood, a closed-form optimal solution does not exist (the cause of difficulty is the sum over k in equation 5.3). A randomized, Expectation-Minimization (EM) based algorithm (Dempster, Laird, and Rubin 1977) will be developed to iteratively search for local optima.

In general, equation 5.8 is only well defined when K is fixed. As such, BoB learns its user-specific model in two stages:

- **Learn Stage I:** Solve for Ω using a fixed K; repeat for each solution in Stage II.
- **Learn Stage II:** Pick the solution from set $\mathcal{T}$:

  $$\mathcal{T} = \Big\{K_{min}\cdots\Omega(\mathsf{K})\cdots K_{max}\Big\}$$

  that looks "best" according to some other fitness measure.

$\Omega(\mathsf{K})$ indicates that K is treated as a variable; alone, Ω implies that K is fixed.

[2] When reporting one of these quantities for an entire dataset, it is first normalized. Dependence on the number of data-points is eliminated by taking the geometric mean.

5.5 Conglomerative Clustering

Conglomerative learning is defined to be the independent modeling of PC, INT, and DIR histograms. With methods `vmnLearn` and `vmnClassify` (defined in Sections 5.7.2 and 5.8), the user's playing modes are learned and perceived as shown in Table 5.2.

learnCGL(X^{C})	classifyCGL($x^{\mathrm{C}}, \Omega^{\mathrm{C}}$)
Ω^{P} = learnVmn(X^{P})	$\left\langle x^{\mathrm{P}}, l^{\mathrm{P}} \right\rangle$ = classifyVmn($x^{\mathrm{P}}, \Omega^{\mathrm{P}}$)
Ω^{I} = learnVmn(X^{I})	$\langle x^{\mathrm{I}}, l^{\mathrm{I}} \rangle$ = classifyVmn($x^{\mathrm{I}}, \Omega^{\mathrm{I}}$)
Ω^{D} = learnVmn(X^{D})	$\langle x^{\mathrm{D}}, l^{\mathrm{D}} \rangle$ = classifyVmn($x^{\mathrm{D}}, \Omega^{\mathrm{D}}$)
return Ω^{C}	$l^{\mathrm{C}} = l^{\mathrm{P}} \cdot l^{\mathrm{I}} \cdot l^{\mathrm{D}}$
	return $\langle x^{\mathrm{C}}, l^{\mathrm{C}} \rangle$

Table 5.2: Conglomerative User-Specific Learning and Classification Algorithms

Independence assumes that:

$$\Pr(x^{\mathrm{C}}) = \Pr(x^{\mathrm{P}}) \cdot \Pr(x^{\mathrm{I}}) \cdot \Pr(x^{\mathrm{D}}),$$

which is why musical surprise, namely labeled likelihood l^{C}, is multiplicative. (The impact of this independence assumption is discussed further in Section 8.5.1.)

5.6 Variable-Sized Histogram Clustering Example

In this section, a representative set of histograms is presented that make BoB's clustering task more concrete. This example demonstrates why clustering histograms with relatively small sample sizes can be non-trivial and why it is beneficial to base inference upon a well-understood probabilistic model.

Twenty-one histograms (M = 12 bins) are displayed in Figure 5.2.[3] Each histogram is a different data-point, generated by the same mixture. What varies between histograms are: how many counts are present and which component in the mixture generated the counts. Given this data, BoB's task is: to determine which value of K to use; and given K, to estimate the model's parameters, Ω, and data-points' clustering, Y.

The first thing to note is that each data-point's sz is known (simply add up the total number of counts), so estimating a distribution for sample size is straight-forward.

[3]This data's generation scheme is described in Section 5.9.1.

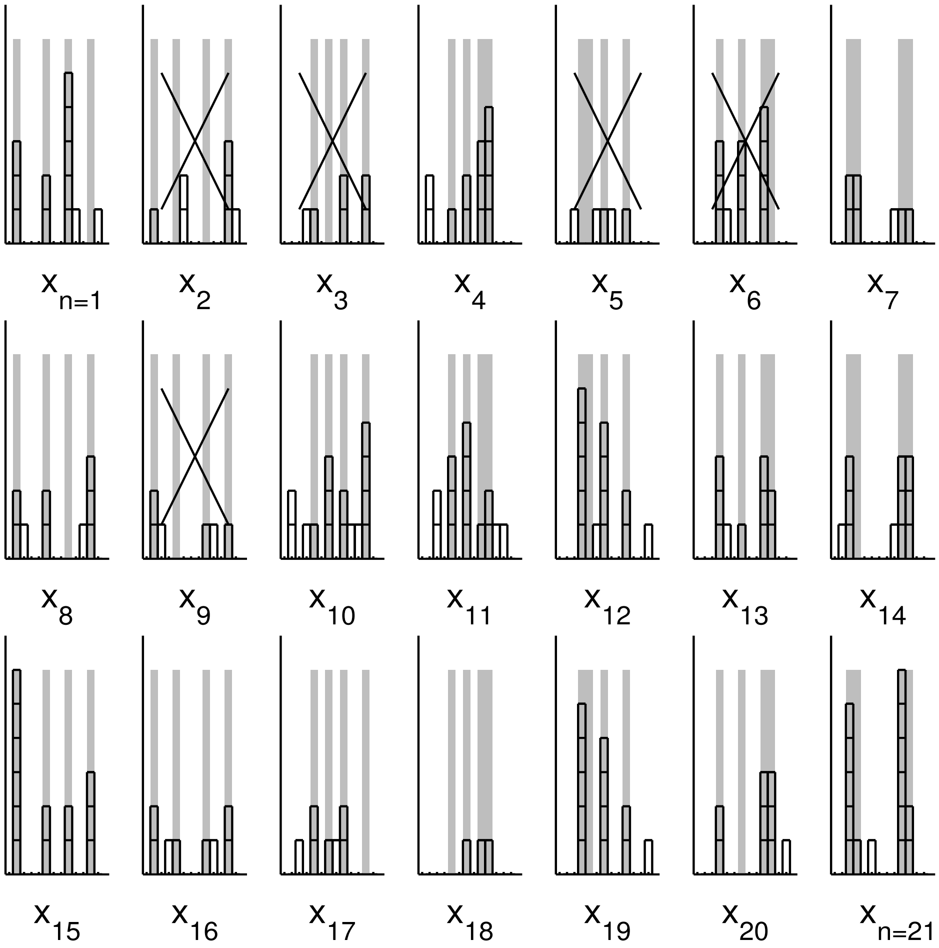


Figure 5.2: A Mixture of Histograms Dataset

Guessing which component generated which histograms is much more difficult. For example, consider trying to cluster this set into $\mathsf{K} = 7$ distinct partitions. The difficulty of this task is exacerbated by the fact that none of the histograms look very much alike, and there are two reasons for this. First, histograms are relatively sparse. Second, the different components' bins "interfere" with one another a "fair bit."

The gray bars in the background identify which bins are most probable given a histogram's underlying component generator. (This knowledge is only available because the data is simulated.) Gray-coincident bins are all equally probable, and all others are equally improbable. Thus, a discerning viewer might notice that the same component was used to generate each column of histograms, e.g., $x_{n=1}$, x_8, and x_{15} belong to some cluster $y = 1$; x_2, x_9, and x_{16} belong to some other cluster $y = 2$;

etc. The gray bars demonstrate what is meant by a "fair bit of interference" among bins. For example, with clusters $y = 1$ and 2, only the 2nd gray bar to the right of the y-axis differs between components. Thus, counts, or a lack thereof, on any other bins do not help discriminate between these two components. In other words, all the other bins *interfere* with one another and are thus non-discriminatory.

Knowing what component each of these histograms belongs to allows several subtle points to be made:

- Often, larger sample sizes contain more useful information. For example, consider x_{15}, where all the counts lie on most probable bins.
- Sometimes larger samples contain a fair bit of misleading information. Consider x_{10}, which has the largest sample size (15 counts), yet $\frac{1}{3}$ of the counts are distributed among improbable bins.
- Smaller samples often contain less reliable information, e.g., consider $\mathsf{n} \in \{2, 3, 5, 6, 9\}$. Each of these data-points is marked with an 'X' because a Bayesian optimal classifier would misclassify them.
- Sometimes smaller samples contain completely pertinent information, e.g., x_{18}.

It is for reasons such as these that a rigorous probabilistic learning procedure is needed — one that weighs larger sample sizes more heavily, especially when they collaborate with other data-points' features.

5.7 Variable-Sized Mixtures of Multinomials

A probabilistic model for clustering histograms that contain an arbitrary number of counts — the *variable-sized mixture of Multinomials* (vMn) — is now derived. Although this derivation was initially motivated by BoB's application, it is completely general and can be applied in any domain where streams of discrete events of arbitrary length need to be structured.

The Multinomial distribution, its incorporation into the vMn model, and the relevant probabilistic quantities are first described, resulting in a complete specification of `classifyVmn`. Next, `learnVmnΩ`, an EM-based learning algorithm for inferring Ω from histogram training set X is presented, resulting in a complete specification of learning Stage I. Numeric quantities for objectively evaluating the quality of learned solution Ω are then derived and used by Stage II to determine how many components to use. These stages will be combined together, producing the complete learning algorithm, `learnVmn`, in Section 5.8.

5.7.1 The Multinomial Distribution

The Multinomial distribution:

$$\text{Mnom}(sz, \theta)$$

extends the Binomial probability distribution to handle more than two discrete, nominal outcomes. Sampling from this distribution is described in Appendix D.1.

The distribution's parameters are interpreted as:

- θ is an M-dimensional probability vector that weights the sides of an M-sided die.
- sz is the fixed number of times that this die is tossed and its value is recorded.

Each of the die's faces are explicitly modeled as independent of one another. Each face is also explicitly treated as nominal — a priori, no two different outcomes are treated as "closer" to one another than any of the other outcomes. For these reasons, the Multinomial is the natural choice for representing a probability distribution over an M-dimensional histogram. However, to handle histograms with a variable number of counts, Mnom must be extended so that sz is a random variable.

With this model, the likelihood of seeing histogram x is simply:

$$\Pr(x|sz, \theta) = sz! \prod_{\mathsf{m}=1}^{\mathsf{M}} \frac{1}{x_{\mathsf{m}}!} (\theta_{\mathsf{m}})^{x_{\mathsf{m}}}. \tag{5.9}$$

The sufficient statistic is the mean:

$$\mathrm{E}_x\left[x_{\mathsf{m}}\right] = sz \cdot \theta_{\mathsf{m}},$$

and with this, variance/covariance are completely specified:

$$\mathrm{E}_x\left[x_{\mathsf{m}} \cdot x_{\mathsf{m}}\right] = sz \cdot \theta_{\mathsf{m}} \cdot (1 - \theta_{\mathsf{m}});$$

$$\mathrm{E}_x\left[x_{\mathsf{m}} \cdot x_{\mathsf{m}'}\right] = sz \cdot \theta_{\mathsf{m}} \cdot \theta_{\mathsf{m}'}.$$

For a discussion of why it is inappropriate to use these statistics to derive a Gaussian approximation of the Multinomial, see Appendix D.2.

5.7.2 The Variable-size Multinomial Mixture Distribution

The variable-sized Multinomial mixture model:

$$\mathrm{vMn}(\eta, \pi, \theta_{1}, \cdots, \theta_{\mathsf{k}}, \cdots, \theta_{\mathsf{K}})$$

comprises K finite Multinomial components, each component using the same random sz variable to control its sample size. From now on, Ω refers to the entire set of vMn parameters.

There are two differences between this mixture model and the general one described earlier:

1. In addition to sampling y (equation 5.1), sz is also sampled from:

$$sz \sim \mathrm{Mnom}(1, \eta).$$

   η is a set of *size weights* operating over the integer range $sz_{min} \leq sz \leq sz_{max}$, indexable via $\eta_{\mathsf{m}=sz}$;

2. Feature vector x (equation 5.2) is replaced with:

$$x \sim \mathrm{Mnom}(sz, \theta_{\mathsf{k}=y}).$$

Under this sampling scheme, the joint probability is:

$$\Pr(x; y|\Omega) = \eta_{sz} \cdot \pi_{\mathsf{k}=y} \cdot \Pr(x|sz, \theta_{\mathsf{k}}),$$

and the last term is defined by equation 5.9. Unlabeled and labeled likelihood become:

$$\mathrm{u}(x|\Omega) = \sum_{\mathsf{k}} \eta_{sz} \cdot \pi_{\mathsf{k}=y} \cdot \Pr(x|sz, \theta_{\mathsf{k}}); \tag{5.10}$$
$$\mathrm{l}(x|y; \Omega) = \eta_{sz} \cdot \Pr(x|sz, \theta_{\mathsf{k}=y}).$$

These equations allow a data-point's posteriors and its classifier (Table 5.3) to be constructed. When classification is based on $\underline{\Omega}$, `classifyVmn` is a *Naive Bayes optimal classifier* (Mitchell 1997) with the Multinomial treating each feature in x independently.

$\boxed{\texttt{classifyVmn}(x, \Omega)}$
$y = \texttt{Argmax}_{1 \le k \le K} \left(\frac{\Pr(x; y=k \mid \Omega)}{u(x \mid \Omega)} \right)$
$l = l(x \mid y; \Omega)$
$\texttt{return}\ \langle y, l \rangle$

Table 5.3: The `classifyVmn` Algorithm

5.7.3 The Sample Size Affect

This section provides insight into how and why sample size influences clustering performance. In Figure 5.3, twenty-five different Mnom distributions over a 2-dimensional feature vector are shown. Each distribution is drawn as a different south-east facing thin line. While each distribution uses the same probability vector, $\theta = \langle 0.75, 0.25 \rangle$, they each have a different sz, ranging from 1 (the line nearest to the origin) to 25 (the line farthest away). The 'o' marks on each line define the input values covered by that distribution. The intersection with the thick black line corresponds to a distribution's mean.

Three different gray regions are drawn on this graph, each enclosing those input values which lie within the central 30%, 70%, and 90% probability mass. Per sample size, the widths of these regions expand via $\mathcal{O}\left(\sqrt{sz}\right)$. At the same time, the total number of possible inputs increases linearly. In other words, as sz increases, there is more and more room for different Mnom components probable regions to not interfere with another. On the other hand, as $sz \to 0$, interference necessarily increases.

5.7.4 Learning a vMn Model from Data

This section considers how to optimize equation 5.8, assuming that a vMn(Ω) model was used to generate the data. Optimizing Ω requires iterative search, and finding a "relatively global" optima requires random restarts. `learnVmnΩ` (Table 5.4) handles both of these issues, using an EM strategy to fit both the training and fake data to an underlying vMn model.

The outer loop (starting at line 1) deals with jumping out of local optima by randomly restarting the search n_{seeds} times. Only that solution, Ω_*, with the lowest unlabelled likelihood, f_*, is returned. (This logic is handled by lines 0a and 8.) Random starts are chosen from the parameter space and are generated by `randomize`. The

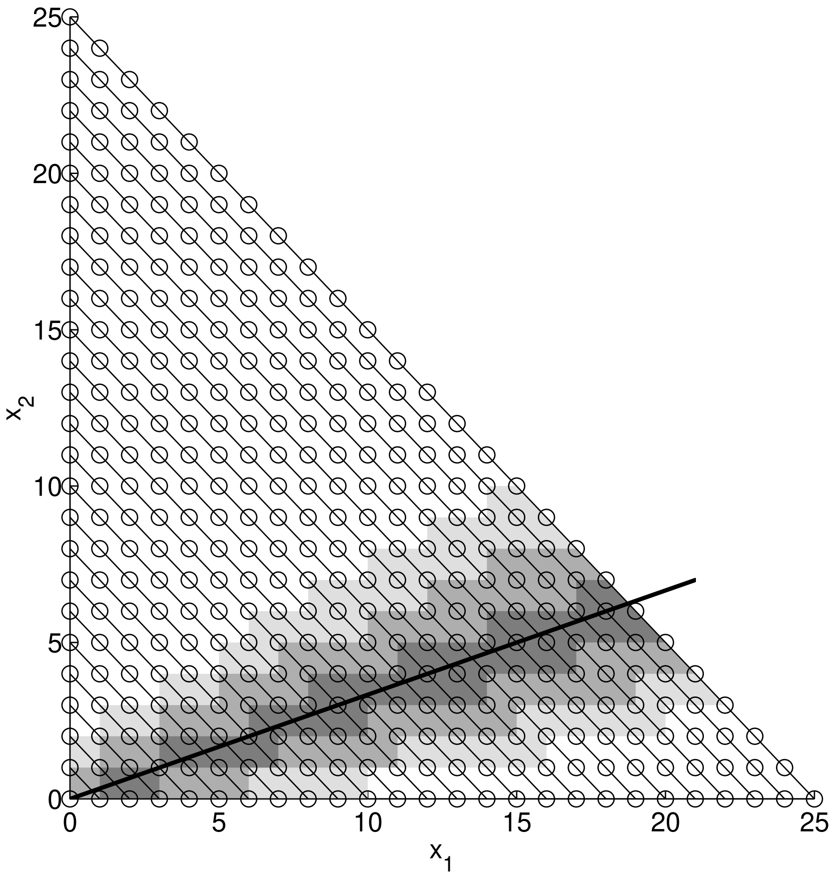


Figure 5.3: Multinomial Dispersion versus Sample Size Example

inner loop (starting at line 2) performs EM, iteratively improving the model's parameter estimate at each time step, `t`. Each of EM's improved solutions and corresponding likelihood fitnesses are indexed via `t`. Variable Ψ holds the training set's posterior vectors, one for each datapoint. vMn specific EM logic is provided in `vMn_E_Step` and `vMn_M_Step`. Finally, lines 6 and 7 introduce early termination conditions.

Randomization

Often, a preliminary clustering algorithm like *k-means* (Duda and Hart 1973) is used to find a good starting point, or *seed*, in which to begin EM (McLachlan and Basford 1988; Smyth 1997). In preliminary tests, no strong correlation between a seed's initial fitness and its final optimized fitness was observed. For this reason, `learnVmnΩ` does not preprocess seeds.

$\boxed{\texttt{learnVmn}\Omega(X^{\boldsymbol{Tr}}, \mathsf{K})}$

0: $\langle X, Y \rangle^{\boldsymbol{Fk}} = \texttt{createPrior}(\mathsf{K})$
0a: $f_* = -\infty$
1: $\texttt{for}\ n = 1;\ n \leq n_{seeds};\ n++$
    $\Omega_{t=1} = \texttt{randomize}(X^{\boldsymbol{Tr}}, \mathsf{K})$
2: $\texttt{for t} = 1;\ \texttt{t} \leq n_{EM};\ \texttt{t}++$
        $f_t = \mathrm{F}(X^{\boldsymbol{Tr}}; X^{\boldsymbol{Fk}} | Y^{\boldsymbol{Fk}}; \Omega_t)$
3: $\Psi_t = \texttt{vMn\_E\_Step}(X^{\boldsymbol{Tr}}, \langle X, Y \rangle^{\boldsymbol{Fk}}, \Omega_t)$
4: $\Omega_{t+1} = \texttt{vMn\_M\_Step}(X^{\boldsymbol{Tr}}, \langle X, Y \rangle^{\boldsymbol{Fk}}, \Psi_t)$
        $f_{t+1} = \mathrm{F}(X^{\boldsymbol{Tr}}; X^{\boldsymbol{Fk}} | Y^{\boldsymbol{Fk}}; \Omega_{t+1})$
5: $\texttt{if}\ \texttt{f}_{t+1} == \texttt{f}_t$
            $\texttt{break}$
6: $\texttt{elseif}\ \texttt{f}_{t+1} < \texttt{f}_t$
            $f_{t+1} = f_t$
            $\Omega_{t+1} = \Omega_t$
            $\texttt{break}$
7: $\texttt{elseif}\ \texttt{f}_{t+1} - \texttt{f}_t < \texttt{f}_{t=1} \cdot \epsilon$
            $\texttt{break}$
        $\texttt{end}$
    $\texttt{end}$
8: $\texttt{if}\ \texttt{f}_{t+1} > \texttt{f}_*$
        $\Omega_* = \Omega_{t+1}$
        $f_* = f_{t+1}$
    $\texttt{end}$
$\texttt{end}$
$\texttt{return}\ \langle \Omega_*, f_* \rangle$

Table 5.4: The `learnVmnΩ` Optimizing Ω Algorithm

`randomize` stochastically chooses seed $\Omega_{t=1}$ as follows:

1. Randomly select K unique datapoints from $X^{\boldsymbol{Tr}}$.
2. Add a small Lagrangian prior to each of these datapoints, normalize them, and use the results to obtain an initial estimate for the Multinomial component parameters.
3. To obtain an estimate for the initial mixture weights, calculate a weighted sum using the distance between each datapoint and the initial component parameters. Distance is Euclidean (datapoints are first normalized).

Expectation

EM performs two analyses per iteration. The first step, `vMn_E_Step` (line 3), calculates how expected the joint likelihood is at time t:

$$\mathrm{E}\Big[\Pr(X^{\boldsymbol{Tr}}; Z^{\boldsymbol{Tr}}|\Omega_{\mathsf{t}}) \cdot \Pr(X^{\boldsymbol{Fk}}; Z^{\boldsymbol{Fk}}; \Omega_{\mathsf{t}})\Big].$$

The training dataset indicator vectors, $Z^{\boldsymbol{Tr}} = \left\langle z^{\boldsymbol{Tr}}_{1}, \cdots, z^{\boldsymbol{Tr}}_{\mathsf{n}}, \cdots, z^{\boldsymbol{Tr}}_{\mathsf{N}} \right\rangle$, are the only random variables. This expectation amounts to simply calculating the posteriors for $X^{\boldsymbol{Tr}}$ at time t:

$$\mathrm{E}\Big[z^{\boldsymbol{Tr}}_{\mathsf{n,k}}\Big] = \psi^{\boldsymbol{Tr}}_{\mathsf{n,k,t}} = \Pr(y_{\mathsf{n}} = \mathsf{k}|x^{\boldsymbol{Tr}}_{\mathsf{n}}, \Omega_{\mathsf{t}}).$$

This entire solution is packaged in Ψ_{t}.

Maximization

The second step, `vMn_M_Step` (line 4), uses the training data posteriors at time t to maximize the model's parameter estimate at time $\mathsf{t}+1$:

$$\Omega_{\mathsf{t+1}} = \mathtt{Argmax}_{\Omega}\Big(\Pr(X^{\boldsymbol{Tr}}; \Psi^{\boldsymbol{Tr}}_{\mathsf{t}}|\Omega) \cdot \Pr(X^{\boldsymbol{Fk}}|Y^{\boldsymbol{Fk}}; \Omega)\Big). \tag{5.11}$$

Solving Equation 5.11 yields:

$$\pi_{\mathsf{k,t+1}} = \frac{\frac{1}{\mathsf{K}} + \sum_{\mathsf{n}} \psi_{\mathsf{n,k,t}}}{1 + \mathsf{N}}, \tag{5.12}$$

$$\theta_{\mathsf{k,m,t+1}} = \frac{\frac{1}{\mathsf{M}} + \sum_{\mathsf{n}} \psi_{\mathsf{n,k,t}} \cdot x^{\boldsymbol{Tr}}_{\mathsf{n,m}}}{1 + \sum_{\mathsf{n}} \sum_{\mathsf{m}} \psi_{\mathsf{n,k,t}} \cdot x^{\boldsymbol{Tr}}_{\mathsf{n,m}}}. \tag{5.13}$$

Interpreting Fake Data as Priors

Equations 5.12 and 5.13 are MAP-based estimates. Parameter estimation is optimal and assumes a Laplacian prior on both π and θ (Vapnick 1982). These priors can be interpreted as introducing fake labeled data. For notational convenience, `createPrior` (line 0) serves as a place holder for simulated fake data.

The prior on θ produces the $\frac{1}{\mathsf{M}}$ term (Equation 5.13). Essentially, this prior assigns one count to each component, distributing it evenly among all bins. In other words, for each component, a fractional labeled data point is placed in $\langle X, Y\rangle^{\boldsymbol{Fk}}$. Given sample size sparsity, the θ priors are especially important because they guarantee that no component's bins ever have a zero probability of occurring. Although this prior makes the learning problem more difficult because the absence of counts on specific bins never completely rules out the probability of a component having generated data, from an improvisational music standpoint, it makes a good deal of sense. Ultimately, this prior guarantees that no specific playing mode is completely unexplorative — unexpected pitch class, interval, and/or directional motion events may always occur.

The prior on π is what produces the $\frac{1}{\mathsf{K}}$ term in Equation 5.12, distributing one extra count evenly among all K components. Once again, this can be interpreted in terms of fractional labeled data, further contributing to $\langle X, Y\rangle^{\boldsymbol{Fk}}$. Since the prior on π encodes the belief that there is always some probability that the user utilizes each playing mode, its musical appropriateness assumes that the musician can generate solos comprised of K different distinctions in real time. Given the small values of K learned for Parker and Grappelli, this assumption is plausible.

Convergence and Termination Conditions

The proof that `learnVmnΩ` converges relies on the fact that the inner iterative loop (line 2) implements EM for vMn(Ω). With each subsequent iteration at time $\mathtt{t}$, EM guarantees that:

$$\mathrm{F}(X^{\boldsymbol{Tr}}; X^{\boldsymbol{Fk}}|Y^{\boldsymbol{Fk}}; \Omega_{\mathsf{t}}) \leq \mathrm{F}(X^{\boldsymbol{Tr}}; X^{\boldsymbol{Fk}}|Y^{\boldsymbol{Fk}}; \Omega_{\mathsf{t+1}}).$$

In theory, finding a local maximum is equivalent to finding a fixed-point, so that:

$$\mathrm{F}(X^{\boldsymbol{Tr}}; X^{\boldsymbol{Fk}}|Y^{\boldsymbol{Fk}}; \Omega_{\mathsf{t}}) = \mathrm{F}(X^{\boldsymbol{Tr}}; X^{\boldsymbol{Fk}}|Y^{\boldsymbol{Fk}}; \Omega_{\mathsf{t+1}}).$$

This ideal fixed point termination condition (line 5) is usually not reached because, in practice, other termination methods are required.

At some point, convergence guarantees are no longer feasible because the computer does not have an unlimited amount of precision (line 6 handles this). Another problem is that EM can take a long time to converge, and how close the current estimate, Ω_{t}, is to a local maxima can not be predicted in advance.[4] `learnVmnΩ` limits search time explicitly, terminating whenever:

[4]John Lafferty, personal communication, May 2001.

- n_{EM} search steps have occurred.
- Per-step improvement has significantly decreased so that the additional change in likelihood has dropped below a small fraction (ϵ, line 7) of the dataset's initial value.

In my experiments, n_{EM} was usually the limiting factor.[5] Since n_{EM}, ϵ, and n_{seeds} directly affect how much time is spent searching for a solution, there is a subtle trade-off between how many times to restart, how long to optimize along a particular path, and how good a solution's fitness is. For this reason, learning results are intimately tied to the *learning setup,* i.e., the specific n_{EM}, ϵ, and n_{seeds} values, that is used.

5.7.5 Assessing A Solution's Quality

When clustering takes place with

> ... no prior information regarding the underlying group structure, or at least where there is no available data from each of the groups if their existence is known... (McLachlan and Basford 1988)

then mechanisms which attempt to understand and quantify a solution's structure and performance become especially useful. Since vMn is model based, one way to quantify Ω's performance is through simulation. Another approach involves repeated re-learning to quantify aspects of the underlying optimization space. The quantification measures presented here are not intended to replace the qualitative analysis of a solution's quality from within the context of its domain (Chapter7).

The measures summarized in Table 5.5 quantify Ω using simulation techniques. Each row outlines a different way of summarizing the quality of Ω. For detailed pseudo-code, see Appendix D.3. Simulation-based procedures rely on the random variable:

$$\langle X, \underline{Y} \rangle \overset{n_{sim}}{\sim} \text{vMn}(\Omega).$$

When reporting a simulation-based quantity, the average value obtained over n_{expt} independent runs is reported. Thus, all of these measures are affected by the values of n_{sim} and n_{expt} that are used. In all but the *hardness* case, simulated data is input to `learnVmnΩ`, so learning setup is also a factor.

[5] This will depend on what value of ϵ was used.

Variable	Quantity	Procedure
hardness	estimates the fraction of inputs that an optimal naive Bayes classifier would misclassify	`estHardness`
cost	estimates the additional fraction of inputs misclassified using a learned (versus optimal) Bayes classifier	`estCost`
overLearn	estimates the amount which learning over-fits the training set's likelihood	`estLearn`
underGeneralize	estimates the amount which learning under-fits the test set's likelihood	

Table 5.5: Simulation Quality Estimates

In a similar fashion, Table 5.6 quantifies Ω by repeating a learning experiment multiple times. Because these procedures do not rely on simulation (instead they use the original training set over and over), their measurements do not rely on n_{sim}. However, because these procedures are stochastic insofar as learning is stochastic, their values are still reported by observing their behavior over n_{expt} independent runs. These distributional results are displayed graphically, either using a histogram or an x-y plot (where averages are connected by a line, and error bars indicate minimum and maximum values). Again, these measures are affected by the value of n_{expt} and the learning setup.

Variable	Quantity	Procedure
instabilities	observes fractions of inputs whose partitions vary between `learnVmnΩ` runs	`obsInstabilities`
fits	observes fitness likelihoods over multiple `learnVmnΩ` runs	`obsFitnesses`
fitDiffs	observes differences between fitness likelihoods that vary between `learnVmnΩ` runs	

Table 5.6: Repeated Learning Quality Estimates

Consistency Relabeling

Since *cost* and *instabilities* compare different inferred clusterings, there is no guarantee that their labelings are consistent, i.e., that their labels rely on the same indexing scheme. This problem arises because, numerically, all possible orderings of K components are identical distributions. By transforming two labelings into a Maximal Bipartite Matching problem (Kozen 1991), Linear Programming techniques (Chvatal 1983) can be used to reorder one cluster's labels so that they map a maximal amount of the training set into the partitions used by another clustering. This remapping is only performed when one or both clusterings are learned.

Hardness

hardness estimates what fraction of the input space is inherently misclassified as a result of vMn's component overlap. In particular, each `estHardness` execution provides an estimate of how many times the optimal Bayesian classifier is expected to be incorrect:

$$\mathit{hardness} \approx \mathrm{E}_x\left[\underline{y} \neq \underline{\hat{y}}\right].$$

This value is approximate because the behavior over the entire input space is approximated by the behavior of n_{sim} data-points.

Learning Cost

cost estimates the additional fraction of inputs that are expected to be incorrect when clustering is inferred using a learned (versus known) set of vMn parameters. The model's inherent hardness is subtracted off from the error rate that results when `learnVmnΩ` provides the clustering:

$$\mathit{cost} \approx \mathrm{E}_X\left[\underline{y} \neq y\right] - \mathrm{E}_x\left[\underline{y} \neq \underline{\hat{y}}\right].$$

This measure is unbiased. Ω is infered from a training set and learning cost is measured using an independent test set. This value is approximate because it is based on n_{sim} data-points. The domain of the first expectation, i.e., X, indicates that this cost is also measured with respect to a specific training set size.

Quantifying Over-Fitting

Simulation can also be used to measure how well — and over-well — Equation 5.8 is optimized during learning. Comparing the training set's true likelihood with its estimated likelihood (*overLearn*) provides a measure of how effective the search for an optimal solution was. At the same time, comparing an independent test set's true likelihood to its estimated value using the learned model (*underGeneralize*) provides a measure of how over-effective the search was. `estLearn` subtracts off the learned estimate from the true likelihood, so the more negative (positive) the *overFit* (*underGeneralize*) value is, the more learning over-fit (under-fit) the training (test) data.

Instability

instabilities records the fractions of inputs that do not map into the same dataset partitions when these partitions are re-learned n_{expt} times. In particular, each pair of solutions[6] is compared to determine how different their partitions are. These values imply a distribution over how repeatable a particular learning task is likely to be, providing some assurance that a specific inferred model is grounded more on actual training set structure than on happenstance.

Observing Likelihood Fitness Variations

fits observes the fitness value obtained each time the training set is re-learned, shedding light on how variable the optimization space's local optima are for a given learning task and setup. *fitDiffs* further analyzes this data, comparing the difference in fitness observed between all pairs of solutions.

5.7.6 Choosing the Number of Components

To fully specify `learnVmn`, an algorithm for Learn Stage II is needed. Recall (Section 5.4.3) that `learnVmnΩ` is repeatedly used to find a set of solutions:

$$\mathcal{T} = \left\{K_{min} \cdots \Omega(\mathsf{K}) \cdots K_{max}\right\},$$

and that Stage II must pick a "best" solution from this set. In the machine learning and statistics literature, this task is typically referred to as *model selection.* This section first discusses some general issues related to model selection, and then the specific approach that BoB will use is described.

Model Selection and Complexity

For vMn($\Omega(\mathsf{K})$), there are $\mathcal{O}(\mathsf{K} \cdot \mathsf{M})$ free parameters to estimate. A particular solution also has $!\mathsf{K}$ equivalent ways of being expressed (simply permute the original

[6] i.e. n_{expt} choose-two pairs in total.

components' ordering). These observations motivate why solution vMn($\Omega(\mathsf{K})$) has *complexity* K.

When a vMn model does not have Lagrangian priors, it can be shown to be part of a general class of learning problems in which the goodness-of-fit function — for the prior-less model, $\mathrm{U}(X^{\boldsymbol{Tr}}|\Omega)$ — is convex within first-order as a function of model complexity (Cadez and Smyth 2001). This result justifies why a well-posed optimization, i.e., one that contains a bounded maximum, necessitates fixing K before Ω is fit to the training set.

Usually, the curves obtained for a Lagrangian prior-based model coincide with this result. Figure 5.4 (left) is typical. In this figure, goodness-of-fit values — for the prior-based model, $\mathrm{F}(X^{\boldsymbol{Tr}}; X^{\boldsymbol{Fk}}|Y^{\boldsymbol{Fk}}; \Omega^{\boldsymbol{Fk}})$ — are displayed for K ranging from 1 to 10.[7] Occasionally, however, BoB would produce a non-concave curve, e.g., Figure 5.4 (right), because this prior-based model does not fulfill all of the conditions used in the Cadez and Smyth (2001) proof. Additional experimentation seems to indicate that concavity violations become more pronounced when the distribution over *sz* becomes smaller.

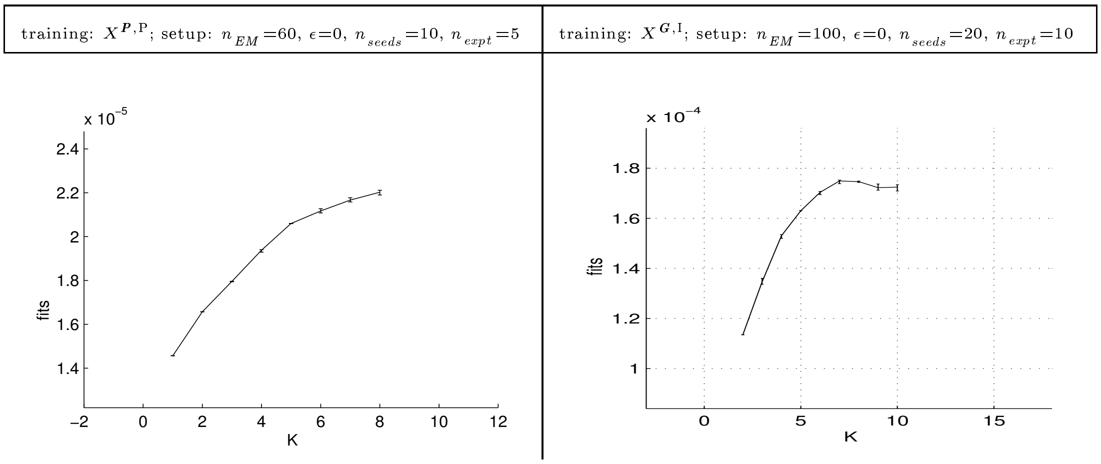


Figure 5.4: Complexity Versus *fits* Examples

Typically, however, fitness tends to increase monotonically. Thus, the shape of BoB's goodness-of-fit function is considered irrelevant when picking K. Rather, an independent heuristic for selecting a "best" solution from set $\mathcal{T}$ is used.

[7]For those values of K in which an error bar is not visible, fitness differed very little between multiple runs.

Complexity and Computation Time

A solution's quality is not only affected by the learning setup, but also by a model's complexity. A more complex model can often achieve a better fitness, but this is usually accompanied by a more variable optimization surface.

One example of this phenomena is presented in Figure 5.5. Each of the nine plots displayed there corresponds to a different complexity, ranging from K=2 (upper left) to K=10 (lower right). Each plot presents a histogram of the *fitDiffs* ddistribution that was observed when learning that particular model (the x-axis scale is 10^{-6}). As complexity increases, the range in variability becomes wider and flatter. What this observation implies is that for a more complex model, one should expect more seeds will be necessary to "adequately" search for a solution.

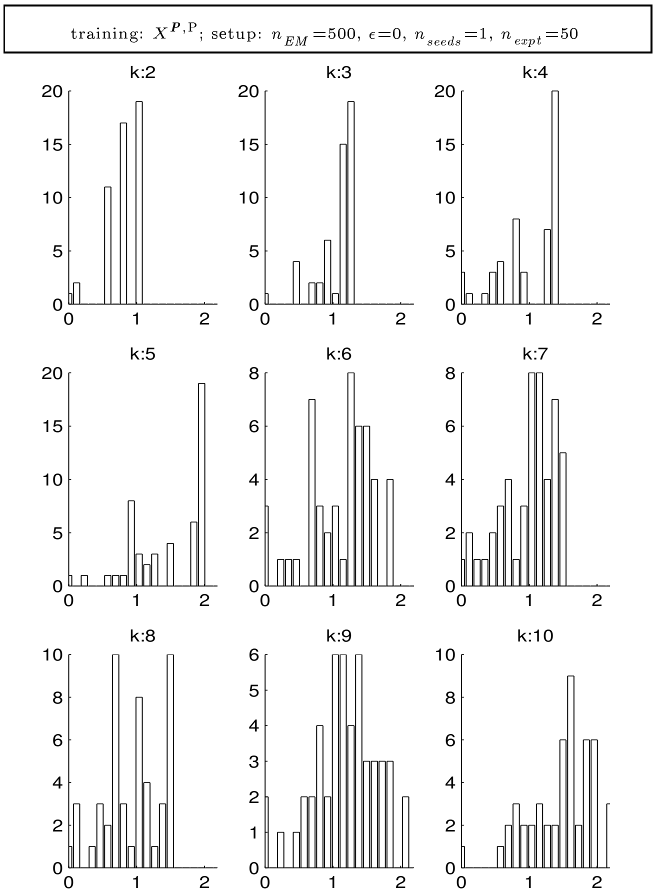


Figure 5.5: Variability in Local Optima Versus Complexity Example

BoB must learn $\Omega(\mathsf{K})$ in a reasonable, fixed period of time. For simplicity, each

$\Omega(\mathsf{K})$ search receives the same learning setup. In other words, regardless of complexity, the same amount of time is allocated for estimating each solution .[8]

Search time can seriously impact learning. For example, the sole cause of the differences between the two *fits* and *instabilities* curves presented in Figure 5.6 is the learning setup (n_{EM}= 60 versus 100, n_{seeds}= 20 versus 10, and n_{expt}= 10 versus 5).

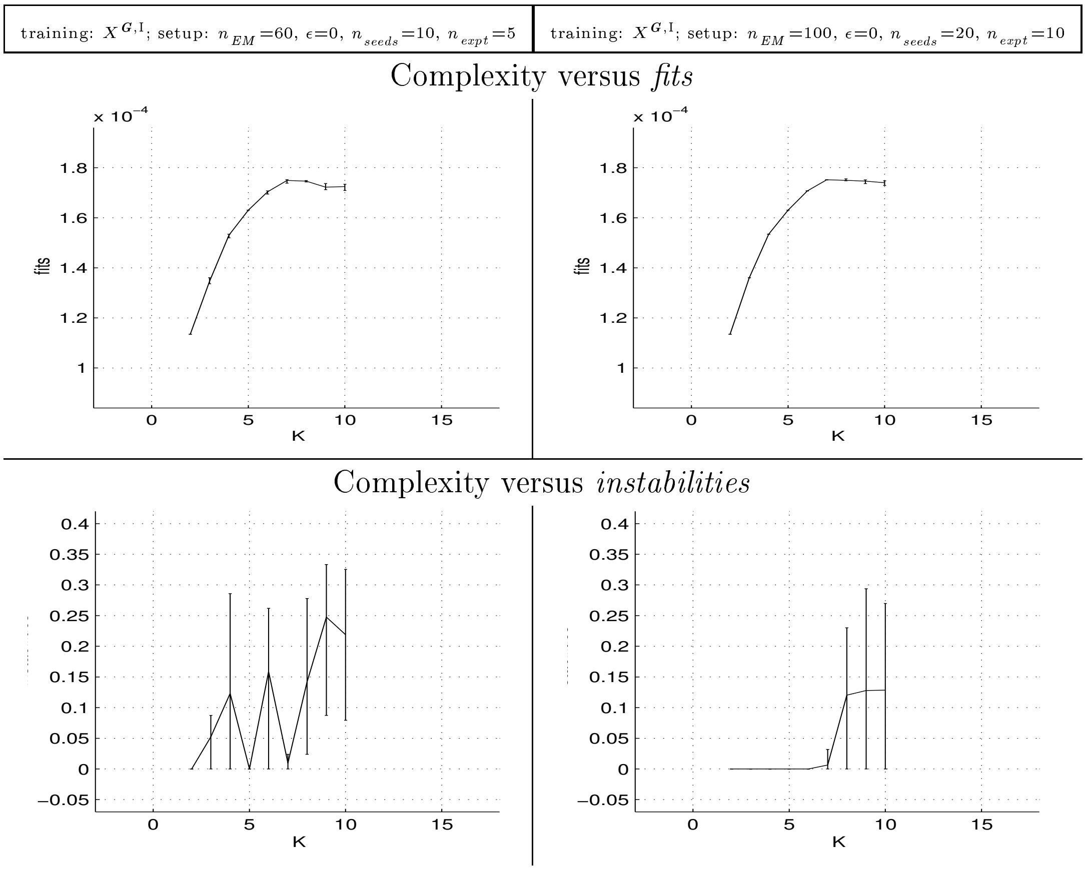


Figure 5.6: Learning Setup and its Affect on Solution Quality Example

A Heuristic For Picking K

One standard technique used in model selection is to add an additional constraint to the goodness-of-fit function, usually incorporating some form of Bayesian or minimum description length penalty term (Mitchell 1997). These penalties are usually on the order of $\mathcal{O}(!\mathsf{K})$. Another common technique employs cross validation, where picking

[8]This statement glosses over that the running time for each iteration of EM is linear with respect to K.

the best component is decoupled from learning Ω. In this case, the goodness-of-fit is based upon an independent test set. Model selection is a very active research area. Rather than concentrating on this very difficult problem, BoB uses a heuristic that is motivated by what is practical for an IMC.

In BoB, three factors contribute to the selection of a "best" $\Omega(\mathsf{K})$:

1. The user's playing modes need to be "real," and realness depends upon how many clusters a particular sparse dataset can support, and how many clusters a particular learning setup can support.

2. Model components should not overlap "too much." From an interactive, adaptive, improvisational point of view, this requirement makes sense — first worry about recognizing and responding to the most prominent shifts in the user's local behavior.

3. It is assumed that, because data is extremely limited, it is better to rely on simulation than cross validation.

These issues are addressed in the following *model selection heuristic*. Pick the largest K for which:

- **Model Selection Condition I**: Instability is no more than 10%;

- **Model Selection Condition II**: Hardness is, on average, no more than 25%.

The first condition deals with "realness," its value inherently linked to the amount of time that was spent learning and the degree to which the training data lends itself to a specific K-partition. This condition is especially conservative, bounding the maximum difference that may be witnessed among any pair of learned partitions, e.g., $\texttt{min}(\textit{instabilities}) < 10\%$. While the 25% rate in Condition II may seem high, BoB is less concerned with guessing every cluster correctly than it is with being able to recognize when a musician switches from one really distinct playing mode to another. In this sense, *hardness* is overly pessimistic, treating all misclassifications as equally bad even though most of these "errors" will have been claimed by very nearby competing partitions.

5.8 The Complete vMn Learning Algorithm

BoB's `learnVmn` is defined in Table 5.7. Ideally, the heuristic just described provides what is needed to specify what `pickBest` does.

$$
\begin{array}{l}
\boxed{\texttt{learnVmn}(X)} \\
\quad \texttt{for}\ \mathsf{k}\ = K_{min};\ \mathsf{k}\ \leq K_{max};\ \mathsf{k}++ \\
\qquad \Omega(\mathsf{k}) = \texttt{learnVmn}\Omega(X, \mathsf{k}) \\
\qquad \mathit{instabilities}_{\mathsf{k}} = \texttt{obsInstabilities}(\Omega(\mathsf{k}), X) \\
\qquad \mathit{hardness}_{\mathsf{k}} = \texttt{estHarndess}(\Omega(\mathsf{k})) \\
\quad \texttt{end} \\
\quad \Omega(\mathsf{K}_*) = \texttt{pickBest}(\mathit{hardness}, \mathit{instabilities}) \\
\quad \texttt{return}\ \Omega(\mathsf{K}_*)
\end{array}
$$

Table 5.7: The Complete `learnVmn` Learning Algorithm

5.8.1 The Number of Components For Parker and Grappelli

The number of components that were learned for Parker and Grappelli are summarized in Table 5.8. Since learning is conglomerative, each model was learned separately. The values observed for each condition are also reported for these solutions. By themselves, these condition values do not indicate why a particular number of components was chosen; one really needs to look at the corresponding hardness and stability curves (Appendix D.4.1 and D.4.2). Parker's PC curves are also shown in Table 5.8 because they are exemplar, a clear "best" model being $\mathsf{K}^{\mathrm{P}} = 5$.

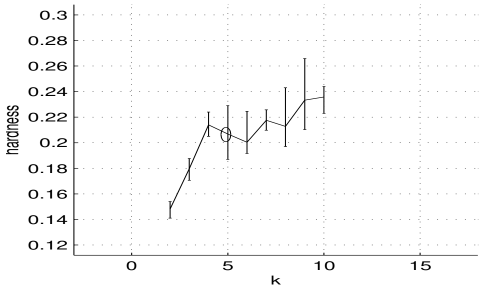
Parker's PC curves were the first that I carefully studied, and their content primed my thinking about what an automated procedure for selecting K should look like. The shape of the *instabilities* curve justifies why `learnVmn` needs to search over a range of Kas opposed to adding one component each time and stopping at the first model for which the conditions fail. In addition, roughly speaking, the *hardness* curve tends to increase as K increases.[9] In light of this tendency, the wording of the heuristic —

[9] I witnessed this trend in all of my vMn modeling experiments. Given the (Cadez and Smyth 2001) proof, which justifies why, for many models, more components provide a better fit to the training data, this trend might seem counter-intuitive. However, when classifying individual data-points, component overlap can only decrease as K increases provided the component's variance can become sufficiently small. The vMn mixture model is constrained by the Lagrangian priors, so their individual component variances will never equal zero. In addition, unlike the Gaussian distribution,

setup: n_{EM}=60, ϵ=0, n_{seeds}=10, n_{expt}=5

Dataset	Number of Components	Condition II	Condition I
$X^{\boldsymbol{P},\mathrm{P}}$	5	0.21	0.01
$X^{\boldsymbol{P},\mathrm{I}}$	4	0.18	0.00
$X^{\boldsymbol{P},\mathrm{D}}$	3	0.06	0.00
$X^{\boldsymbol{G},\mathrm{P}}$	4	0.14	0.01
$X^{\boldsymbol{G},\mathrm{I}}$	5	0.15	0.00
$X^{\boldsymbol{G},\mathrm{D}}$	4	0.05	0.09

Parker's PC *hardness* curve



Parker's PC *instabilities* curve

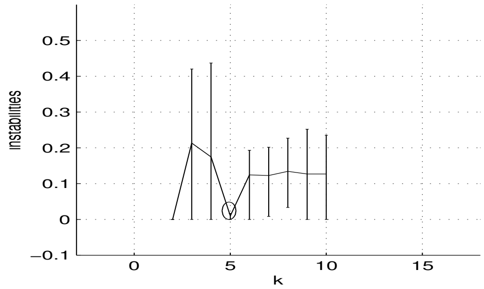


Table 5.8: Parker and Grappelli Number of Components Results

'pick the largest K' — can be interpreted with respect to picking the largest number of clusters for which the clustering is still "distinct enough" to be useful.

For all the models that were chosen, only a relatively small number of components were stable enough to pass Condition I, and in all cases, this was the limiting factor. Intuitively, it makes sense that a musician would only employ a small number of modes when spontaneously generating melodic material. Excluding $X^{\boldsymbol{G},\mathrm{D}}$, these models are also extremely stable, almost all pairs of solutions identically partitioning the training set.

5.8.2 Caveats

Unfortunately, the `learn`vMn algorithm is not the end of the clustering story. Questions remain:

---

the Multinomial's variance is completely specified by its mean, θ, and its sample size, sz. It is with respect to labeled likelihood that fitness improves with increased K, because more components allow data-points to lie closer to their estimated means. The relationship between component overlap, sz, K, and training set size warrants further study.

- What values, K_{min} and K_{max}, should be used to bound the search?
- Should the search only increment k by one each time (rather than performing, for example, some form of binary search)?
- What learning setup and n_{expt} values should be used to produce "good enough" *hardness* and *instabilities* measures?

About one year ago, after performing numerous simulations, relearning parameters several times, and investigating their quality from a musical standpoint, I settled upon the number of components reported in Table 5.8. It was not until recently, after rescrutinizing the curves presented in Appendix D, that I realized that this model selection heuristic would have chosen different solutions for the Grappelli PC and INT models, (choosing $\mathsf{K}^{\boldsymbol{G},\mathrm{P}} = 10$ versus 4, and $\mathsf{K}^{\boldsymbol{G},\mathrm{I}} = 7$ versus 5).

At this point, I decided to relearn the Grappelli INT model again, to see if, with a more exhaustive search regime, the curves that result would eliminate this discrepancy. During this investigation I became intimately familiar with the dependence of a solution's quality assessment on computation time (this investigation led to the curves in Figure 5.6). Although with Condition I, the more exhaustive search would still have chosen $\mathsf{K}= 7$ (Figure 5.6, right), it is easy to see why the jaggedness in the original curve (left) distracted me from selecting the "best" heuristic solution.

With Grappelli's PC model, it seems unlikely that Grappelli, a much less chromatic player than Parker, would use twice as many PC modes to drive his improvisations than Parker did. The shape of his PC *instabilities* curve also supports the notion that the $\mathsf{K}= 10$ model is suspect. In particular, this curve seems to imply that at some point K becomes "large enough" for the partitioning to again become reliably reproducible.[10] This leads me to wonder what other results might change had I, for example, used $K_{max} = 20$ versus 10 in my experiments.

The main point I have learned from this experience is that it is unreasonable to assume that a hard and fast model-selection heuristic will always provide an uncontested "best" result, and there is no reason to believe that several different models may not be equally acceptable. In addition, while model selection is an important issue, computationally, it is expensive to investigate. This is especially the case with Conditions I and II, which require that n_{expt} different learning experiments be run for

[10] The relationship between instability, sz, K, and training set size also warrants further study.

all values of K. In a commercially viable system, additional heuristics must be used to reduce this search space.

As the models in Table 5.8 became the basis for BoB's stable implementation, the results reported in the rest of this thesis use these models.

5.9 Performance

The performance of the vMn is empirically validated in two ways. First, how well `learnVmnΩ` handles sparsity is investigated. This investigation uses a higher-level model from which specific vMn parameterizations are obtained. Next, to assess these solutions' quality, simulations using the parameterizations learned for Parker and Grappelli are conducted.

5.9.1 With Respect Sparsity

Three different representative sparse datasets were generated to investigate how the performance of `learn`vMn is affected by sparsity. Because these datasets are simulated, their underlying structure is known, which makes it easy to assess how well `learn`vMn can learn them.

As presented in Table 5.9 (lefthand columns), each dataset is controlled by two high-level meta-parameters, α and α' (defined shortly). Each dataset is referred to by its scenario index, e.g., X_{I} is the simulated dataset generated using $\alpha' = 0.248$ and $\alpha = .001$. Various learning assessments are recorded in the middle and far right columns.

setup: n_{EM}=500, ϵ=10^-8, n_{seeds}=10, n_{sim}=175, n_{expt}=25

Simulation			Misclassifications			Fitness	
Scenario	α'	α	*hardness*	*cost*	total error rate	*overLearn*	*underGeneralize*
I	0.248	.001	0.11	0.00	0.11	-0.21	0.18
II	0.21	.020	0.22	0.07	0.29	-0.30	0.38
III	0.188	.031	0.31	0.16	0.47	-0.34	0.40

Table 5.9: Simulated Dataset Performance

For a given scenario, data is sampled from vMn as follows:

1. Use seven equally likely components.

2. Create $\mathsf{N} = 175$ data-points (this produces, on average, 25 histograms per cluster).

3. Sample sz from a uniform distribution that spans 3 to 15 counts.

4. The Mnom components are defined using meta-parameters α and α':

$$\begin{aligned}
\theta_1 &= \langle \alpha, \alpha, \alpha, \alpha, \alpha, \alpha, \alpha, \alpha, \alpha, \alpha', \alpha', \alpha', \alpha' \rangle \\
\theta_2 &= \langle \alpha', \alpha, \alpha, \alpha, \alpha, \alpha, \alpha, \alpha, \alpha, \alpha, \alpha', \alpha', \alpha' \rangle \\
\theta_3 &= \langle \alpha', \alpha', \alpha, \alpha, \alpha, \alpha, \alpha, \alpha, \alpha, \alpha, \alpha, \alpha', \alpha' \rangle \\
\theta_4 &= \langle \alpha', \alpha', \alpha', \alpha, \alpha, \alpha, \alpha, \alpha, \alpha, \alpha, \alpha, \alpha, \alpha' \rangle \\
\theta_5 &= \langle \alpha', \alpha', \alpha', \alpha', \alpha, \alpha, \alpha, \alpha, \alpha, \alpha, \alpha, \alpha, \alpha \rangle \\
\theta_6 &= \langle \alpha, \alpha', \alpha', \alpha', \alpha', \alpha, \alpha, \alpha, \alpha, \alpha, \alpha, \alpha, \alpha \rangle \\
\theta_7 &= \langle \alpha, \alpha, \alpha', \alpha', \alpha', \alpha', \alpha, \alpha, \alpha, \alpha, \alpha, \alpha, \alpha \rangle .
\end{aligned}$$

   Each component strongly prefers four bins with probability α'. The rest are fairly improbable, occurring with probability α. The location of each component's α and α' bins where chosen to provide as much intra-component interference as possible, so that combinations of features are often required to discriminate between components.

Data generation for scenario II was presented in Figure 5.2.

The only difference between these datasets is what values of α and α' are used. These values control how much individual component probability distributions overlap with one another. As a result, in Table 5.9, the *hardness* estimate increases as α' decreases. In other words, by increasing α', individual components become more and more alike, so particular features become less and less useful for discrimination.

The *cost* column supports the hypothesis that, as inherent hardness increases, so does the penalty associated with having to learn Ω. The relation between *hardness* and *cost* is explained in part by the corresponding trends in fitness. As hardness increases, overlap in the training set components provides more chance for over-fitting (*overFit* decreases). This over-fitting makes an independent test set look overly unlikely (*underFit* increases).

The total error rate expected for a particular learned classifier is the sum of *cost* and *hardness*. While these rates may look high, keep in mind that a random classifier will, on average, misclassify 86%, so learning definitely produces useful feedback about the model's underlying structure. This error rate is also bounded from below in terms of *hardness* — no algorithm will, on average, misclassify less than this amount.

It is reassuring to see that, with an 11%-hard training set, learning accrued no additional cost. Similarly, the 22%-hard dataset accrued only 12 additional mismatches (7%), for a grand total of 29% incorrect mappings. In part, model selection Condition II was chosen because learning begins degrading at this hardness value.

These results are based on simulated data that is as sparse as a representative PC dataset. Thus, it is reasonable to expect similar results when learning from a musician's PC or INT data. This claim is somewhat optimistic because, in these simulations, the number of components is known in advance and a vMn mixture model generated the data. On the other hand, with real musician data, one might expect a less contrived mixture (in these simulations, components were chosen to interfere as much as possible).

5.9.2 With Respect To Parker And Grappelli

To further assess quality of the Parker and Grappelli learned models presented in Section 5.8.1, these parameters were use to generate new simulated datasets.

As presented in Table 5.10 (lefthand columns), each simulation is identified by its specific scenario index, e.g., X_{I} is the dataset obtained by independently sampling 119 data-points from $\mathrm{vMn}\left(\Omega^{\boldsymbol{P},\mathrm{P}}(5)\right)$. Various learning assessments are recorded in the middle and far right columns.

setup: $n_{EM}=60$, $\epsilon=1,\cdot 10^{-8}$, $n_{seeds}=10$, $n_{expt}=25$

Simulation			Mismatches			Over-fitting	
Scenario	Parameter	n_{sim}	*hardness*	*cost*	total error rate	*overLearn*	*underGeneralize*
I	$\Omega^{\boldsymbol{P},\mathrm{P}}(5)$	119	0.21	0.11	0.32	-0.31	0.30
II	$\Omega^{\boldsymbol{P},\mathrm{I}}(4)$		0.18	0.13	0.31	-0.28	0.19
III	$\Omega^{\boldsymbol{P},\mathrm{D}}(3)$		0.06	0.03	0.09	-0.32	0.25
IV	$\Omega^{\boldsymbol{G},\mathrm{P}}(4)$	126	0.13	0.05	0.18	-0.20	0.17
V	$\Omega^{\boldsymbol{G},\mathrm{I}}(5)$		0.15	0.08	0.23	-0.22	0.20
VI	$\Omega^{\boldsymbol{G},\mathrm{D}}(4)$		0.04	0.03	0.07	-0.35	0.40

Table 5.10: Musician Clustering Performance

In all cases, *cost* contributes less to total error than *hardness* does. Cost is most damaging when learning Parker's INT model, and least damaging with the DIR models. The main reason why DIR models suffer little is because there is so little overlap among their various components. When compared with a random classifier, the usefulness of learning models for Parker and Grappelli are apparent. For example, while the Parker PC model can be expected to misclassify 32% of its data, this is significantly better than misclassifying 80%.

5.10 Discussion

5.10.1 Probabilistic Models

The probabilistic vMn scheme is model-based, which means that, with a learned model, one can:

1. map new data-points into their most likely underlying generators;
2. estimate likelihoods;
3. generate new data.

The mathematical, probabilistic underpinnings of this model also mean that its assumptions and underlying basis are completely specified. With this specification, an immediate wealth of techniques — e.g., EM and its guarantee of convergence, probabilistic reasoning, Bayesian online updating, etc. — are immediately available for the borrowing. While others in the computer music community have applied more ad-hoc techniques for music clustering with some success (Rolland and Ganascia 1996; Pennycook and Stammen 1993), others have recently been calling attention to the benefits of using more formal, probabilistic methods in music domains (Raphael 2001; Cemgil and Kappen 2001a).

My appreciation of this formality was a long time in coming, and as a result I can attest to to how much easier my current, EM-based approach was to implement than my previous, less principled approach was. Originally, I tried to use a *k-medians* clustering algorithm (Kaufman and Rousseeuw 1990), with a χ^2 measure (Press, Teukolsky, Vetterling, and Flannery 1992) as the pairwise similarity between histograms. This measure was especially attractive because it is probabilistically defined to, in the limit, measure how likely it is that two histograms come from the same underlying distribution. One of the problems with this measure, however, is that it is not a distance metric (i.e., it does not necessarily satisfy the triangle inequality). This is part of why it was unclear what update mechanism should be used to ensure that, per-iteration, a cluster's means would always improve the clustering result. After several months, I got a χ^2 median-based algorithm working, but when I ran it on real music data, I ran into an additional problem: difficult-to-reason about infinite loops. In contrast, an EM-based solution is guaranteed to converge (Dempster, Laird, and Rubin 1977) and minimize the Kullback Leibler distance between the model and the underlying distribution that generated the data (Kearns, Mansour, and Ng 1997).

In addition, although the χ^2 measure does have a probabilistic interpretation, it was not directly obtained from a generative model, which makes it less useful to an IMC. More generally, in artistic domains, where objectivity is so difficult to measure, a formal probabilistic model provides an additional foundation on which to investigate performance. In the case of Bob, this is done from both from a machine learning point of view (Section 5.9) and from a musical one (Chapter 7).

5.10.2 Markov Chain Clustering

Multinomial clustering and its application to the inference of Markov chains provides a powerful mechanism for inferring structure in sequences, and has received some recent attention in the AI community (Thom 2000b; Ramoni, Sebastiani, and Cohen 2000b; Ramoni, Sebastiani, and Cohen 2000a; Sebastiani, Ramoni, and Cohen 2000; Smyth 1999; Cadez, Gaffney, and Smyth 2000).

From a machine learning point of view, the most exciting aspect of the vMn model is that, in addition to clustering variable-length data sequences, empirically, it has produced reasonable results when data is "relatively sparse." The key insight is that a histogram's generative component can define a Markov chain (as was done with Ω^{D} and Figure 4.5). vMn not only clusters sequences, but also infers their underlying transition matrixes, by which new, similarly distributed sequences can be sampled.

5.10.3 Sparsity

Sample size sparsity and its relation to histogram dimensionality is hard to quantify. Higher histogram dimensions mean that more probable bins may exist, which provides more opportunities for bin interference. On the other hand, larger dimensions provide more possibilities for different components' probable bins to *not* interfere with one another.

Consider the interplay between dimensionality and sparsity in Figure 5.7. Both the histograms for the melodic segment shown in Figure 5.1 and the underlying component parameters belonging to its estimated conglomerative class are displayed. x^{D} is the most sparse, with 27-dimensions and only five counts. Roughly speaking, eight bins in θ^{D} are non-negligible (marked by a '*') so there are a fair number of possibilities for which this generator could have placed these counts into the bins of x^{D}. Similarly, there are seven non-negligible bins in θ^{I} (marked by a '$\times$') and θ^{P} (marked by a '$\circ$'), yet each of these histograms contains six counts. In considering this simple example, one can imagine that the same component could generate a wide variety of different looking histograms. It is for this reason that the generalization scheme provided by vMn is so rich.

One way to view sparsity is as a mechanism through which particular histograms only reveal *partial information* about which of their underlying generator's features are probable. In this light, one can consider using sparsity/partial information to facilitate the simulation of creative participation in live musical exchanges. For example, one way to interpret "creative" behavior is with respect to how much information about a generator is revealed in a given context.

5.10.4 Generality

So far the focus of vMn has been on learning structure from relatively short and sparse pitch-sequence segments. However, the vMn technique is a completely general methodology for modeling structure in any sequence composed of nominal, discrete symbols. Provided that an appropriate set of symbols is available in which to build histogram feature vectors, this same algorithm could be used to find structure in other musical sequences, e.g., rhythms, chords, etc. More general applications also exist. For example, one promising idea would be to use vMn to automatically analyze log files on a computer, so that structure about various failure states might be learned.[11]

[11] This possibility was first suggested to me by Marko Grobelnik.

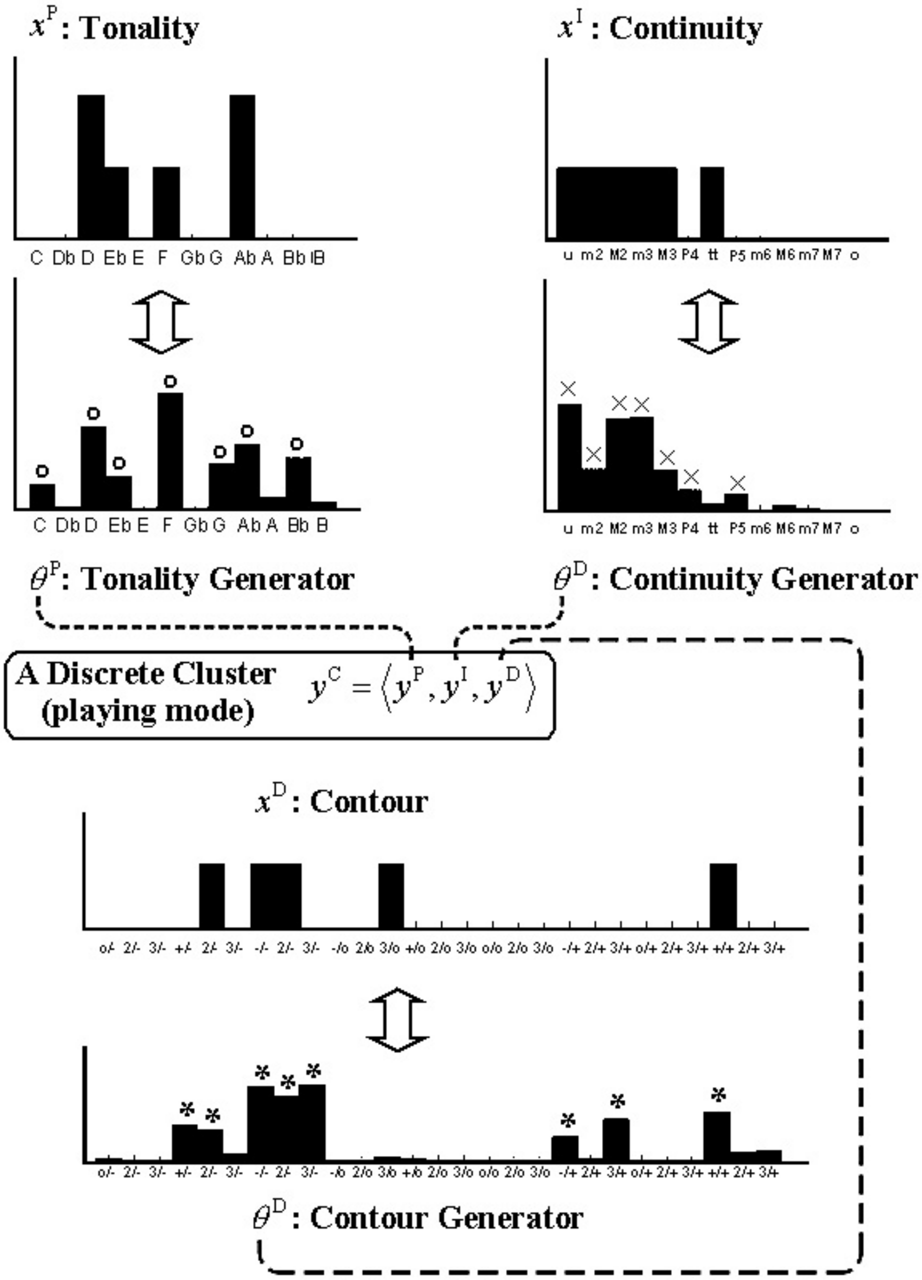


Figure 5.7: A Conglomerative Clustering Example

5.10.5 Musical Surprise

One subtle but important domain-specific decision was made when building the `classifyVmn` algorithm (Table 5.3). It might seem curious that this function returns an estimate's labeled likelihood versus its posterior, especially given that the posterior quantifies how certain the algorithm is about its clustering estimate. I decided on this approach because labeled likelihood only increases when a data-point lies closer to its estimated component's mean, whereas the posterior increases anytime a feature vector moves away from its competing clusters' decision boundaries.

I decided that it was more important for BoB to be able to understand when the musician was realizing one of their playing modes in a very surprising manner than it was to recognize when it was very certain that a particular playing mode estimate was correct. In musical terms, I interpret the estimated labeled likelihood as how *surprising* the music appears (the smaller the likelihood, the more surprising). Notions of surprise or expectation has received a fair amount of attention in music perception and analysis communities, e.g., (Bharucha 1994).

5.10.6 Musical Modeling Assumptions

It is worth considering the underlying assumptions that are made when a vMn model is used to learn about a solo's structure from its conglomerative feature vectors.

It is easy to interpret PC components as musical scales (e.g., Table 7.1), and from this context, problematic assumptions become immediately apparent (similar issues arise with the INT and DIR models). Assumptions that are made when clustering PC histograms include:

	Assumption:	**Violation/Feature?**	**Cause:**
(i)	Within a given bar, one pitch class does not affect or depend on any other.	Strong Violation: from a scalar standpoint, pitches, and their ordering in time, do influence one another.	The Multinomial assumes feature independence.

(ii)	Per bar, the musician randomly flips a weighted die to determine which scale they will use.	Medium Violation: scales in surrounding bars can set a context that influences this choice.	A Multinomial is used to sample the mixture weights.
(iii)	Any form of relationship or ordering between pitch-class bins is explicitly not assumed.	Feature: this scenario lets the data itself determine which pitch classes should be viewed as more alike.	Multinomial components are, by definition, nominal and discrete.
(vi)	Segments that contain more leaves contain more information about their underlying scales.	Feature: it is often easier to determine the tonality in a local segment of melody when it contains a fast run of notes.	Histograms are not normalized by their sample size.

The remaining assumptions are discussed within the context of conglomerative clustering:

	Assumption:	**Violation/Feature?**	**Cause:**
(v)	The number of leaves per segment does not depend upon playing mode.	If this assumption is violated, I expect its affect is small.	Sample size and mode are treated independently.
(vi)	The number of leaves in different bars does not affect or depend on one another.	Mild Violation: how many notes are played in surrounding bars could influence this choice.	The Multinomial is used to sample η.
(vii)	Pitch class, intervallic, and directional events do not influence each other within a bar.	Strong Violation: melodically, pitches, intervals, and directions, and their ordering in time, do influence one another.	First, the conglomerative learning assumption means that PC, INT, and DIR models are treated independently. Second, see Cause (i).

The most egregious violations, (i), (ii), and (vii) are all due to unreasonable independence assumptions. However, these assumptions produce simpler models, which are less susceptible to over-fitting. It has been recognized by other ML researchers that Naive Bayes classifiers, which tend to focus on decision boundaries, are often fairly robust to independence assumption violations (Nigam, McCallum, Thrun, and Mitchell 1998). In BoB, my hope is that inference on a smaller, more relevant dataset will compensate for the model's simplifying assumptions.[12]

The most egregious melodic violations are assumptions (i) and (vii). Melodies are temporal phenomena, and what comes before affects what comes after. Fortunately, the conglomerative view does capture a fair amount of temporal structure. One reason that I am not too concerned about this issue is that the clusters learned for Parker and Grappelli produce musically reasonable, musician-specific results (Chapter 7). In addition, in the generative model, additional temporal constraints can be introduced via clamping (Section 6.3.5).

Musically speaking, assumption (iii) is most interesting because it provides a mechanism for letting the melodies themselves determine which pitch class (interval or direction) bins are closest to each other. In particular, within a given mode, those bins that share identical probabilities are viewed as equivalent, and the closer that two bins probabilities are, the more similarly their counts will be treated. Given the higher-level structure of the mixture model, this becomes a very powerful musical idea because it allows the data itself to drive similarity in a context-sensitive way.

[12] In this sense, these violations become features.

Chapter 6

Generating Musician-Specific Response

This chapter introduces two inter-related stochastic mechanisms: rhythmic transformation and goal-driven pitch sequence generation. Together, these mechanisms are used by BoB to transform a call into a novel, user- and context-specific solo response in real time.

Rhythmic transformation tweaks a call's rhythm, recursively modifying its VLT to produce a response's rhythm. Since this approach operates upon a VLT's hierarchy, it directly benefits from the VLT's musically powerful rhythmic encoding. This transformation is non-trivial because it must ensure that any tree it produces is also a VLT.

The values of the pitched leaves in the response rhythm are generated using a goal-driven pitch-sequence sampling scheme. In particular, a sequence that maps into a particular *goal* — i.e., exhibits a specific user playing mode — is sought. The difficulty in this task is efficiently integrating what was learned from the user-model into a coherent sampling scheme. This task must also accommodate a fairly general set of constraints, so that what is generated can seamlessly integrate into the environment, with various kinds of domain knowledge and/or user preferences, etc. This scheme's performance is evaluated with respect to how "well" it simulates new datasets for Parker and Grappelli.

6.1 Overview

As outlined in Figure 3.3, generation takes place in the context of call and response. In particular, the call's context in bar r is used to drive the generation of a novel, user-specific response in bar r'.

Generation occurs in three stages:

1. **Generate Stage I:** Tweak the internal structure of call rhythm v_{r}, producing response rhythm, $v_{\mathsf{r}'}$.
2. **Generate Stage II:** For the pitch-valued leaves in $v_{\mathsf{r}'}$, generate a new pitch sequence, $\zeta_{\mathsf{r}'}$, that:
   (a) incorporates the very most recent local context, $mrp_{\mathsf{r}'-1}$;
   (b) mimics the call's user-specific perception, y^{C} and l^{C}, mapping back into the same playing mode with the a similar amount of musical surprise;
   (c) can meet a fairly general set of optional constraints.
3. **Generate Stage III:** Assign sequence $\zeta_{\mathsf{r}'}$ to the pitched leaves of $v_{\mathsf{r}'}$ and playback the result.

With this method, a response is "similar" to its call in that the former's rhythm is a stochastic transformation of the latter. With respect to conglomerative trends, similarity relies on the fact that both, at least in principle, are generated by the same underlying playing mode distribution.

The primary focus in this chapter is Stage II, which, because of sample-size sparsity, can provide significant generalizations in the agent's behavior.

6.2 Rhythmic Transformation

In this section, a stochastic algorithm that recursively tweaks a VLT's internal structure is presented.

6.2.1 Node-Specific Data Types

To describe how a VLT is tweaked, some high-level implementation details are needed. Since various VLT nodes contain different types of information, VLTs are implemented as non-homogeneous data structures. Polymorphic behavior is maintained (across classes, methods behave in a class-specific manner). Three node classes are used:

- `leaf` **class** contains either a pitch or a rest value. Method `isRest` distinguishes between these two. Only when this method returns `true` can fields `pitch` and `tie` be queried.
- `int2` **class** contains two child nodes, a left child, `l`, and a right child, `r`.
- `int3` **class** contains three child nodes, a left child, `l`, a middle child, `m`, and a right child, `r`.

Internal nodes are comprised of classes `int2` and `int3`. When all of the children in an internal node are leaf nodes, it is an *internal terminal node.* This condition is identified by method `isTerminal`. The *fringe* of a VLT is defined to be all the tree's leaf and internal terminal nodes.

6.2.2 Rhythmic Embellishment and Simplification

Because of the VLT's hierarchical nature, techniques for embellishing and simplifying rhythm are straightforward. Consider first *embellishment*, where a part of the VLT's fringe is expanded (Figure 6.1, left). *Simplification,* on the other hand, involves collapsing part of the VLT's fringe (Figure 6.1 right).

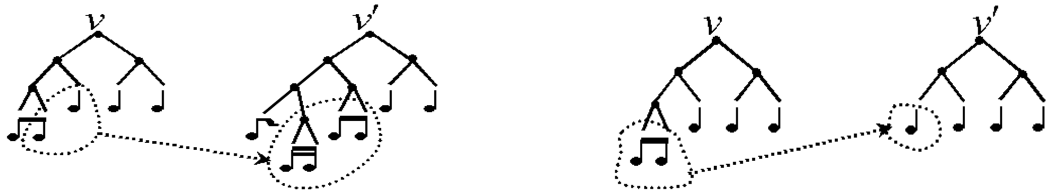


Figure 6.1: Rhythmic Embellishment (left) and Simplification (right)

6.2.3 Maintaining the Properties of a VLT

The internal structure of a VLT was carefully restricted to accommodate only encodable rhythms. When tweaking a tree's internal structure, care must be taken to ensure that the result produced is also a VLT (i.e., the rhythmic restrictions in Chapter 4 must remain invariant). Restrictions I and II are easy to maintain: simply disallow subdivisions or mergers that produce a node (or nodes) whose duration(s) are not valid.

A *degenerate* tree violates Restriction III, and therefore is not as small as possible. Avoiding degeneracy is more difficult and accounts for most of the complexity associating with transforming a rhythm. A degenerate tree branch and its unique VLT equivalent are shown in Figure 6.2. The dashed line links back into an arbitrary, entire tree, whereas the specific branch being discussed is drawn with solid black lines. The dotted arc that connects the two notes (left) indicates that these leaves form a compound duration (they would be tied during playback). As such, this branch has an equivalent, smaller representation (right). The key observation to take away from this example is that, provided Restriction III is maintained, an entire set of leaf siblings will never contain identical values, for if they did, a simpler tree would have been used to construct them.[1]

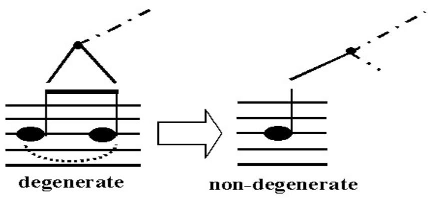


Figure 6.2: A Degenerate Tree Branch and its Corresponding VLT

The pitch sequence generator in Stage II handles degeneracy by disallowing unison intervals at key locations within the sequence. Degeneracy handling is simplified by maintaining the the following invariants during rhythm tweaking:

- **Tweak Invariant I:** When all of a node's siblings are leaf values, at least one of them must not become a rest.
- **Tweak Invariant II:** Rests can only be modified in the rhythmic transformation stage.

Part of this degeneracy-handling strategy is outlined in Figure 6.3, where leaves are drawn as triangles. The white triangle is a rest value, and the black and gray triangles are arbitrary, non-identical pitch values. In v, pitched leaves are separated by a rest, so degeneracy will never result, even if both have the same pitch value.

[1]This statement assumes that the leaves comprising composite durations are not considered identical.

Thus, the interval between these two pitches is irrelevant, i.e., the zig-zag line which links them is marked with a 'T' (`true`). Locations where the unison interval must be disallowed are marked with 'F' (`false`).

Now consider trees v' and v''. Across sibling boundaries, identical pitches are never degenerate, so unison intervals are allowed. With Invariants I and II, it is sufficient to disallow intervals only in those internal terminal nodes whose children are all pitches. It is for this reason that the 'F' is placed on the `int3` node of v' and the `int2` node of v''. The only remaining possibility, which concerns Invariant I, is addressed in Section 6.2.6.

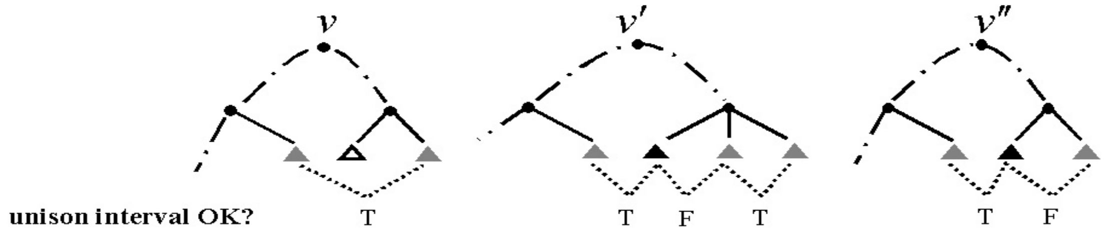


Figure 6.3: Avoiding Degeneracy: Constraining where Unison Intervals Occur

6.2.4 Node-Specific Tweak Methods

The `recursiveTweak` methods (Table 6.1) are used to transform a VLT, starting at the root node and proceeding, in order, through its descendents. Only those nodes in the fringe are possibly modified using the `fringeTweak` methods (Table 6.2). The stochastic parameters that control fringe tweaking (the subscripted 'a' variables) are presented in Section 6.2.5.

Internal fringe nodes are possibly modified by the calls in lines 1 through 4 of `recursiveTweak`. If modification does occur, it takes place in lines 2 through 4 of `fringeTweak`. Fringe nodes are tweaked by first sampling a random number generator (line 1) and then determining what change (if any) this sample corresponds to. The `isValid` check (Section 4.3.1) ensures that rhythmic modifications produce valid results. In each of these methods, the last outcome is ignored because it is a *no-op* (no change is made). The `nonRestChild` logic is explained in Section 6.2.6.

For details concerning class-specific notation, see Appendix A. In addition, when `this` is assigned, it is assumed that a node's type can be changed within its modifying method and that garbage collection automatically takes care of itself.

```
leaf :: recursiveTweak()
1: this.fringeTweak()

int2 :: recursiveTweak()
     if this.isTerminal()
1:     this.fringeTweak()
     else
       if this.l.isLeaf()
2:       this.l.fringeTweak()
       else
         this.l.recursiveTweak()
       end
       if this.r.isLeaf()
3:       this.r.fringeTweak()
       else
         this.r.recursiveTweak()
       end
     end
```

```
int2 :: recursiveTweak()
     if this.isTerminal()
1:     this.fringeTweak()
     else
       if this.l.isLeaf()
2:       this.l.fringeTweak()
       else
         this.l.recursiveTweak()
       end
       if this.m.isLeaf()
3:       this.m.fringeTweak()
       else
         this.m.recursiveTweak()
       end
       if this.r.isLeaf()
4:       this.r.fringeTweak()
       else
         this.r.recursiveTweak()
       end
     end
```

Table 6.1: `recursiveTweak` Methods

leaf :: fringeTweak()

```
-1: if this.isRest()
      return
0:  end
    p = this.pitch; t = this.tie
    d = this.duration
    ω = ⟨a_{1→2}, a_{1→3}, a_1⟩
1:  outcome = Mnom(1, ω)
    if outcome == 1 and isValid(1, 2, d)
2:    this = int2
        (leaf(p, d/2, t), leaf(p, d/2, t))
    elseif outcome == 2 and isValid(1, 3, d)
3:    this = int3
        (leaf(p, d/3, t), leaf(p, d/3, t), leaf(p, d/3, t))
    end
```

int2 :: fringeTweak()

```
    child = this.nonRestChild()
    p = child.pitch; t = child.tie
    d = child.duration
    ω = ⟨a_{2→1}, a_{2→3}, a_{2→x}, a_2⟩
1:  outcome = Mnom(1, ω)
    if outcome == 1 and isValid(2, 1, d)
2:    this = leaf(p, 2·d, t)
    elseif outcome == 2 and isValid(2, 3, d)
3:    this = int3(leaf(p, d·2/3, t),
        leaf(this.m.pitch, d·2/3, this.m.tie),
        leaf(this.r.pitch, d·2/3, this.r.tie))
    elseif outcome == 3
4:    this = int2(this.l.fringeTweak(),
        this.r.fringeTweak())
    end
```

int3 :: fringeTweak()

```
    child = this.nonRestChild()
    p = child.pitch; t = child.tie
    d = child.duration
    ω = ⟨a_{3→1}, a_{3→2}, a_{3→x}, a_3⟩
1:  outcome = Mnom(1, ω)
    if outcome == 1 and isValid(3, 1, d)
2:    this = leaf(p, d·3, t)
    elseif outcome == 2 and isValid(3, 2, d)
3:    this = int2(leaf(p, d·3/2, t),
        leaf(this.r.pitch, d·3/2, this.r.tie))
    elseif outcome == 3
4:    this = int3(this.l.fringeTweak(),
        this.m.fringeTweak(),
        this.r.fringeTweak())
    end
```

int2 :: nonRestChild()

```
    if !this.l.isRest()
      return this.l
    else
      return this.r
    end
```

int3 :: nonRestChild()

```
    if !this.l.isRest()
      return this.l
    elseif !this.m.isRest()
      return this.m
    else
      return this.r
    end
```

Table 6.2: `fringeTweak` Methods

6.2.5 Stochastic Rhythm Parameters

Table 6.3 displays the parameters that control how likely various elaboration and simplification tweaks are. The parameters in the upper part control `leaf` node transformations; the middle and bottom parts affect `int2` and `int3` transformations respectively.

The subscripts on the probabilities indicate what transformations these variables control. The field to the left of the '$\rightarrow$' refers to the node class upon which the transformation is performed (1, 2, and 3 corresponding to nodes `leaf`, `int2`, and `int3`). The field to the right of the arrow refers to what class this node becomes should the transformation take place. When this latter field is an 'x', the node's own type doesn't change, but any or all of its children's types may change. The last subscript (no arrow) corresponds to the no-op.

Effect	Transform	Probability	Value
growth	leaf into 2 children	$a_{1\rightarrow 2}$	$\frac{\gamma}{2(\gamma+\gamma'')}$
growth	leaf into 3 children	$a_{1\rightarrow 3}$	$\frac{\gamma}{2(\gamma+\gamma'')}$
none	no-op	a_1	$\frac{\gamma''}{(\gamma+\gamma'')}$
collapse	2 child leaves into parent leaf	$a_{2\rightarrow 1}$	$\frac{\gamma'}{\gamma+\gamma'+\gamma''}$
growth	2 child leaves into 3	$a_{2\rightarrow 3}$	$\frac{\gamma}{2(\gamma+\gamma'+\gamma'')}$
growth	transform left and right child leaves	$a_{2\rightarrow x}$	$\frac{\gamma}{2(\gamma+\gamma'+\gamma'')}$
none	no-op	a_2	$\frac{\gamma''}{(\gamma+\gamma'+\gamma'')}$
collapse	3 child leaves into parent leaf	$a_{3\rightarrow 1}$	$\frac{\gamma'}{2(\gamma+\gamma'+\gamma'')}$
collapse	3 child leaves into 2	$a_{3\rightarrow 2}$	$\frac{\gamma'}{2(\gamma+\gamma'+\gamma'')}$
growth	transform left, middle, and right child leaves	$a_{3\rightarrow x}$	$\frac{\gamma}{(\gamma+\gamma'+\gamma'')}$
none	no-op	a_3	$\frac{\gamma''}{(\gamma+\gamma'+\gamma'')}$

Table 6.3: Fringe Modification Parameters

The stochastic modification scheme presented here is simple because no higher-level reasoning determines how much growth, collapse, and stasis is desirable in a given context. Rather, the following set of meta-parameters are set before on-line interaction takes place, and these fully determine how likely various fringe tweaks will be:

- **embellishment factor**, γ;
- **simplification factor**, γ';

- **no-op factor**, γ'';

6.2.6 An `int3` Node Modification Example

Figure 6.4 illustrates all the possible decision points in `int3` :: `fringeTweak`. The top-most '•' is the original `int3` terminal node, upon which `int3` :: `fringeTweak` is called. Its left, middle, and right-most children are drawn in order underneath as the black, white, and gray triangles (for the meaning of these triangles, see Section 6.2.3). Inside `int3` :: `fringeTweak`, one of four decisions will be made, depending upon the sampled value, *outcome*. Each decision is drawn as a thick gray arrow, along with its accompanying likelihood.

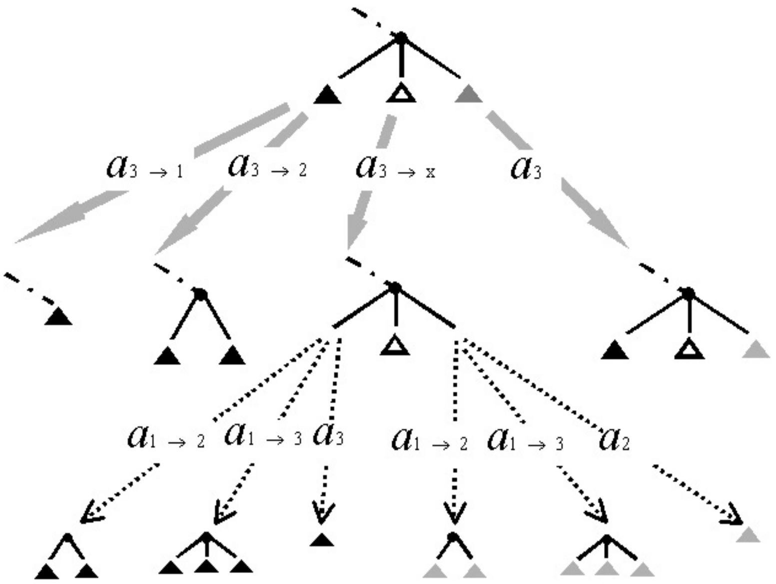


Figure 6.4: `int3` :: `fringeTweak` Decision Points

The first decision (leftmost gray arrow) considers collapsing `int3` into a single leaf. This action will only take place if: outcome $a_{3\to 1}$ is selected, and it is valid to triple the node's duration. If this node is collapsed, the pitch of its left-most child (as determined by `nonRestChild`) is used as a place holder for the new node.

As long as `nonRestChild` returns a non-rest child, which child it selects is unimportant. (This child's content is merely a pitch-valued place-holder whose final value is determined in Stage II.) Thus, the decision to have `nonRestChild` scan from left to right, returning the first pitched child that is found (in this example, the black triangle) is arbitrary. This place-holder must not be a rest because, as justified in Figure 6.5, if both `int3` and its sibling were allowed to become rests, degeneracy could result.

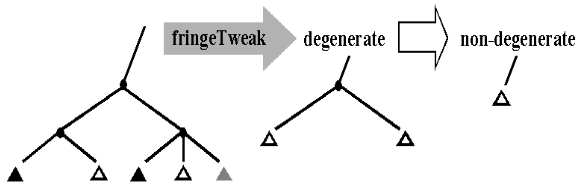


Figure 6.5: Rest Degeneracy Example

The second decision (gray arrow $a_{3\to 2}$) considers collapsing the `int3` terminal node into an `int2` terminal node. Again, only valid transformations are allowed and non-degeneracy requires that at least one of the placeholder children is not a rest.

Rather than changing `int3` directly, the third decision (gray arrow $a_{3\to x}$) attempts to transform each of its child leaves. The dotted lines emanating out from these children indicate what decision points are possible when `leaf` :: `fringeTweak` is called on them. Note that leaf tweaking can only embellish (grow) a rhythm. Invariant II is the reason why the middle child (a rest) remains untouched (line -1 in `leaf` :: `fringeTweak`). At first glance, it may appear that the $a_{3\to x}$ transformation is not really necessary — after all, `int3` :: `fringeTweak` has already been called. However, if this decision point were removed, there would be no way for an `int3` node to grow. Furthermore, this mechanism allows complete VLT branches to become unbalanced, and such trees can be rhythmically desirable.

The last decision (gray arrow a_3), is a no-op.

6.3 Goal-Driven Pitch-Sequence Generation

In this section, an algorithm is developed for generating an arbitrary length pitch sequence that simultaneously:

1. tends to exhibit a specific *goal,* i.e., user playing mode y^{C};
2. meets a fairly general set of optional constraints.

Goal-driven generation provides the crucial mechanism for closing the loop between learning to perceive and being able to respond "in kind." The

constraints ensure that what is generated can seamlessly integrate into the local environment, higher-level domain knowledge, user desires, etc.

A key technical issue is to merge the (perhaps conflicting) desires of the PC, INT, and DIR trends implied by y^{C} into a coherent pitch sequence sampling scheme. This issue is first considered in terms of an *exact solution* — one in which all three goal trends are simultaneously guaranteed to asymptotically occur — because it sets the stage for the more practical approach, `generatePS`, that BoB uses. The *practical approach* is simpler than the exact solution because it assumes that PC, INT, and DIR trends can be treated independently. However, the practical approach also considers a more difficult task because it guarantees that certain external constraints are met.

The practical solution is appropriate for real time because its independence assumption provides an *efficient* sampling scheme, e.g., computation time is polynomial in terms of the number of inputs. While this assumption produces sequences that tend to exhibit properties of the desired goal, y^{C}, sequences are not asymptotically guaranteed to do so. The odds of generating a sequence that exhibits the desired conglomeration of trends exponentially increases as the `generatePS` distribution is randomly re-sampled. Function `searchPS` performs this higher-level search, first re-sampling and then defining a fitness measure that specifies which item from a set of sequences is "best."

6.3.1 Conglomerative-Driven Generation

How does one develop a generative algorithm that combines the perhaps conflicting trends required of different temporal views, $y^{\mathrm{P}}, y^{\mathrm{I}}$ and y^{D}, into a coherent pitch sequence sampling scheme? This task is difficult because:

- PC trends entirely ignore pitch ordering.
- INT trends only care about adjacent pitch pairs.
- DIR trends care about varying-length subsequences.

At a high level, it is useful to think of generation in terms of reversing the arrows in Figure 5.1. However if generation merely reversed the training set's mapping, only snippets that the musician has already played could be retrieved. What is really needed is the ability to generalize — to be able to generate new solos that, when perceived using `classifyVmn`, map back into the same playing mode. *Conglomerative generation* is defined as this process of generating a new pitch sequence, ζ, of arbitrary length T:

$$\zeta = \langle \zeta_1, \cdots, \zeta_t, \cdots, \zeta_{\mathsf{T}} \rangle$$

that maps back into conglomerative goal, y^{C}.

Ideally, the knowledge contained in user-model $\mathrm{vMn}\left(\Omega^{\mathrm{C}}\right)$ could be used to define a randomized pitch-sequence sampling scheme. Towards this end, the shorthand:

$$\theta^{\mathrm{C}} = \left\langle \theta^{\mathrm{P}}, \theta^{\mathrm{I}}, \theta^{\mathrm{D}} \right\rangle,$$

refers to the goal's *relevant components.* θ^{P} is the y^{P}-th component of Ω^{P}, θ^{I} is the y^{I}-th component of Ω^{I}, and so on.

6.3.2 Why the Obvious Approach Does Not Work

It is reasonable to consider using θ^{C} directly to generate a new set of histograms:

$$\begin{aligned} x^{\mathrm{P}} &\sim \mathrm{Mnom}(\mathsf{T}, \theta^{\mathrm{P}}) \\ x^{\mathrm{I}} &\sim \mathrm{Mnom}(\mathsf{T}, \theta^{\mathrm{I}}) \\ x^{\mathrm{D}} &\sim \mathrm{Mnom}(\mathsf{T}, \theta^{\mathrm{D}}) \end{aligned}$$

This sampling scheme, however, is unhelpful because it does not explain *how* to arrange some selection of pitches so that all of the sampled PC, INT, and DIR counts are accounted for.

For example, consider the two histograms in Figure 6.6. There is no way to choose and order two pitch values such that:

- one of them uses pitch class D♭ and the other E♭;
- the interval between them is either a tritone or a perfect 4th.[2]

While Ω^{C} is useful for abstracting higher-level trends from a solo by ignoring various aspects of a pitch sequence's temporal order, it is precisely for this reason that the generative model does not necessarily produce a viable solution when sampling from it.

[2]The left over interval would have to connect with the MRP in a previous segment.

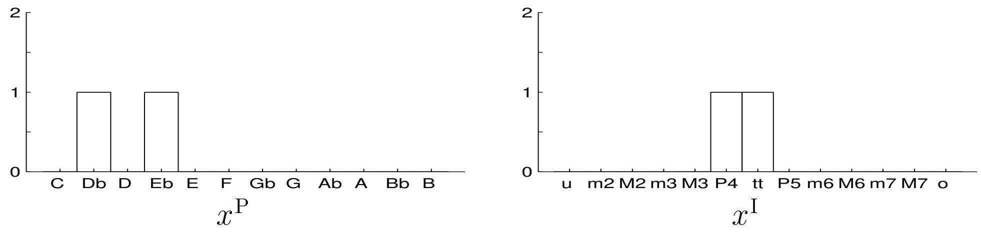


Figure 6.6: A Pair of Histograms That Do Not Produce a Pitch Sequence

6.3.3 Stochastic Processes

Randomness must be introduced to reverse the many-to-one mapping, $\zeta \longrightarrow y^{\mathrm{C}}$, that conglomerative generative requires. A *stochastic process* that operates over pitch values provides an ideal construct for reversing this mapping because of its explicit:

- randomness;
- handling of temporal order;
- mechanisms for affecting how likely various pitch sequences will be.

For more details about stochastic processes, see (Ross 1993). What follows is a brief synopsis of that reference.

Consider stochastic process:

$$\left\langle q_{1}, \cdots, q_{\mathsf{t}}, \cdots, q_{\mathsf{T}} \right\rangle$$

that runs for a finite period of time, T, and transitions between a finite set of states:

$$\forall\ \mathsf{t},\quad q_{\mathsf{t}} \ \in\ \mathcal{S}\ = \{1, 2, \cdots, \mathsf{s}, \cdots, \mathsf{S}\}.$$

Furthermore, suppose that whenever this process is in state $q = \mathsf{s}$ there is a fixed probability, $\Lambda_{\mathsf{s},\mathsf{s}'}$, that it will next transition into state s' in the next time step. In particular:

$$\forall\ \mathsf{t},\quad \Pr(q_{\mathsf{t}+1} = \mathsf{s}' | q_{\mathsf{t}} = \mathsf{s}) = \Lambda_{\mathsf{s},\mathsf{s}'}.$$

This stochastic process is a *Markov Chain* because the probability of transitioning from one state into another is only conditioned on the current state.

A Markov Chain's *state transition matrix*, Λ, must adhere to the following constraints:

$$\forall\ \mathsf{s}, \mathsf{s}',\quad \Lambda_{\mathsf{s},\mathsf{s}'} \geq 0; \tag{6.1}$$

$$\forall\ \mathsf{s}',\quad \sum_{\mathsf{s}} \Lambda_{\mathsf{s},\mathsf{s}'} = 1. \tag{6.2}$$

By assuming an ergodic, irreducible Markov Chain, additional equations that summarize its long-term behavior are obtained. In particular, there exists a unique *stationary probability distribution*, λ, defined over states so that:

$$\frac{\#\left\{1 \leq \mathsf{t} \leq \mathsf{T} | q_{\mathsf{t}} == \mathsf{s}\right\}}{\mathsf{T}} \overset{\mathsf{T}\to\infty}{\longrightarrow} \lambda_{\mathsf{s}},$$

or equivalently:

$$\forall\ \mathsf{s}',\quad \sum_{\mathsf{s}} \lambda_{\mathsf{s}} \cdot \Lambda_{\mathsf{s},\mathsf{s}'} = \lambda_{\mathsf{s}'}; \tag{6.3}$$

$$\forall\ \mathsf{s},\quad \lambda_{\mathsf{s}} \geq 0; \tag{6.4}$$

$$\sum_{\mathsf{s}} \lambda_{\mathsf{s}} = 1. \tag{6.5}$$

Provided Λ is known, straightforward matrix algebra can be used to solve for λ.

6.3.4 An Exact Solution

An exact solution is obtained by deriving a Markov chain that:

- operates over a set of states comprised of pitch and relevant history pairs;
- provides the long-term behavior required of θ^{C}.

Provided that a long enough walk was sampled from this exact chain, the resulting pitch sequence could be transformed into histogram, x^{C}, that, when normalized, would converge to θ^{C}. How long convergence would take depends upon the chain's *settling time*, which in turn depends upon how many times matrix Λ must be multiplied with itself before it reaches a fixed point. How long a sampled sequence must be before its histogram has enough counts to map into conglomerative goal y^{C} is, while related, a different question, whose answer depends to some extent upon how hard the underlying mixture model is.[3] Certainly for BoB's short segmentation windows, long-term guarantees become less relevant. Nonetheless, the exact Markov Chain is derived because it provides a good basis for understanding the conglomerative generation problem.

[3]Recall that a clustering becomes less distinct as the number of counts decreases.

Definition

An exact solution is parameterized by: θ^{C}, p_{min}, and p_{max}. The latter two parameters specify a consecutive set of semi-tones:

$$\mathcal{P} = \left\{p_{min}, p_{min} + 1, \cdots, p_{max} - 1, p_{max}\right\},$$

from which pitch sequence ζ is sampled.[4] $\mathcal{P}$ contains $n_{\mathcal{P}} = p_{max} - p_{min} + 1$ elements. The set of relevant histories (Section 4.4.6) is:

$$\mathcal{H} = \{+, ++, +++, o, oo, ooo, -, --, ---\},$$

and so $n_{\mathcal{H}} = 9$. The exact Markov Chain operates over a tuple composed of both sets:

$$\mathcal{S} = \left\{\langle p, h\rangle | p \in \mathcal{P};\ h \in \mathcal{H}\right\}.$$

Notation $\mathsf{s} \in \mathcal{S}$ and $\mathsf{s}' \in \mathcal{S}$ are used interchangeably to refer to tuples $\langle p, h\rangle$ and $\langle p', h'\rangle$ respectively. Thus, for example, wherever s' appears, p' and h' are implied (and vice versa).

An exact chain's transition probabilities must satisfy equations 6.1 and 6.2. Furthermore, its stationary distribution must exist, satisfying equations 6.3 through 6.5. Asymptotic convergence to θ^{C} is assured with some additional equations:

1. **PC constraints** ensure that, with a long enough sequence, the distribution over pitch classes converges to θ^{P}:

$$\sum_{p \in \left\{i \in \mathcal{P} | \mathtt{pc}(i) == \mathsf{m}\right\},\ h \in \mathcal{H}} \lambda_{\mathsf{s}} = \theta^{\mathrm{P}}_{\mathsf{m}}. \tag{6.6}$$

   $\mathtt{pc}$ was defined in Section 4.4.3.

2. **INT constraints** ensure that, with a long enough sequence, the distribution implied by adjacent pitches' absolute intervals converges to θ^{I}. Specifically, when absolute interval m is larger than the unison interval:

$$\sum_{\mathsf{s} \in \mathcal{S}} \lambda_{\mathsf{s}} \cdot \Lambda_{\mathsf{s},\mathsf{s}'=<p+\mathsf{m},\mathtt{hs}(h,+)>} + \sum_{\mathsf{s} \in \mathcal{S}} \lambda_{\mathsf{s}} \cdot \Lambda_{\mathsf{s},\mathsf{s}'=<p-\mathsf{m},\mathtt{hs}(h,-)>} = \theta^{\mathrm{I}}_{\mathsf{m}}. \tag{6.7}$$

   When m is equal to the unison interval:

$$\sum_{\mathsf{s} \in \mathcal{S}} \lambda_{\mathsf{s}} \cdot \Lambda_{\mathsf{s},\mathsf{s}'=<p,\mathtt{hs}(h,o)>} = \theta^{\mathrm{I}}_{\mathsf{m}}. \tag{6.8}$$

   $\mathtt{hs}$ and $\mathtt{sym}$ were defined in Sections 4.4.5 and 4.4.6.

[4] In the context of pitch, '+' and '−' represent addition and subtraction by semi-tones.

3. **DIR constraints** ensure that, with a long enough sequence, the transitions out of relevant history states (Figure 4.5) converge to θ^{D}:

$$\sum_{\mathsf{s} \in \mathcal{S},\ p' \in \left\{i \in \mathcal{P} | \mathtt{sym}(p,i)==j\right\}} \lambda_{\mathsf{s}} \cdot \Lambda_{\mathsf{s},\mathsf{s}'=<p',\mathtt{hs}(h,j)>} = \theta^{\mathrm{D}}_{\mathtt{m}=\mathtt{dInd}(h,j)}. \qquad (6.9)$$

`dInd` was defined in Section 4.4.6.

This exact solution extends the Markov Chain in Figure 4.5 so that, instead of having states defined just over $\mathcal{H}$, it now considers all possible pairs between $\mathcal{P}$ and $\mathcal{H}$. Whereas Figure 4.5 had nine states, this chain has $n_{\mathcal{S}} = n_{\mathcal{P}} \cdot n_{\mathcal{H}}$. Similarly, instead of three output transitions per state, there are now $n_{\mathcal{P}}$, one for each possible next pitch value. However, the underlying directional history updating structure is the same, and reproducing structure in this way provides the same benefits (fewer parameters to fit, a sparser transition matrix). The difference is that now, instead of each transition corresponding only to a direction, it also corresponds to a specific pitch. With these states, it is easy to produce random pitch sequences that asymptotically converge to specific conglomerative trends. Simply take a random walk, strip away the history part of the tuple, and report the sequence of pitches left over.

An Example

A single step in a Markov Chain operating over the range $p_{min} = \mathrm{C}:4$ to $p_{max} = \mathrm{E}:4$ is displayed in Figure 6.7 (octave values are not shown). This chain operates over $n_{\mathcal{P}} = 5$ pitches. Its complete graph, $n_{\mathcal{S}} = 45$, has five different pitches on which to transition out of any given state. Even with this small $n_{\mathcal{P}}$, the entire graph for one step is too big to display. Figure 6.7 only contains those states that are necessary for transitioning out of a state that is currently on pitch D : 4, $\mathsf{s} \in \{\mathcal{S} | p = \mathrm{D}:4,\ h \in \mathcal{H}\}$. Dotted arcs correspond to downwards motions, solid arcs to upwards motions, and dot-dashed arcs to unison intervals.

Properties of an Exact Solution

An exact solution requires solving for both Λ and λ, but pairs of these variables appear as products in some of the equations. If either Λ or λ were known, then solving for the other would reduce to a set of linear equations with greater-than-zero constraints. Linear Programming could be used to solve for the unknown variables. With pairwise

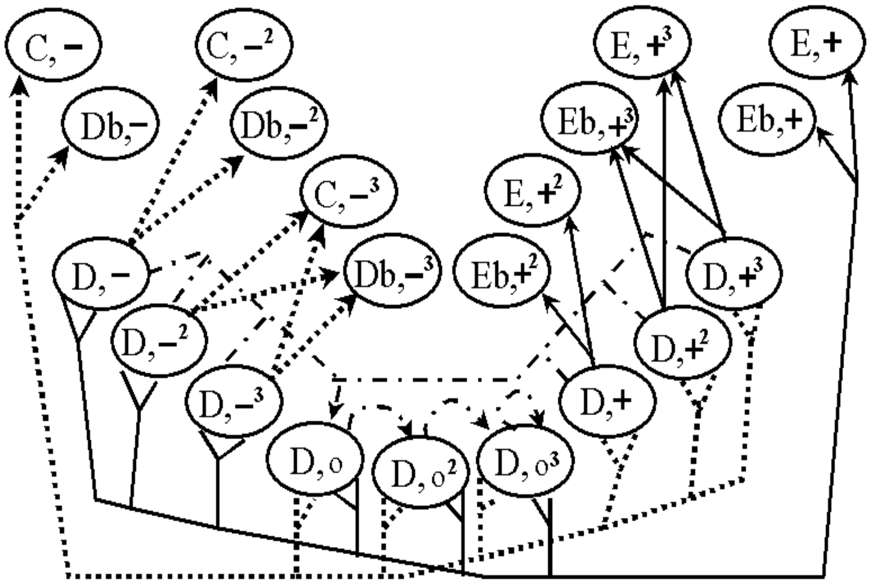


Figure 6.7: One Step in Part of an Exact Markov Chain

products, however, a general solution requires Quadratic Programming, an NP-hard task.

To solve for Λ, $n_{\mathcal{P}} - 1$ parameters per state must be estimated, resulting in a total of $n_{\mathcal{S}} \cdot (n_{\mathcal{P}} - 1)$ free variables. λ introduces an additional $n_{\mathcal{S}} - 1$ parameters, six of which, because of implicit bounds implied by $\mathcal{P}$, are zero. In total, an exact solution requires $9 \cdot n_{\mathcal{P}}^2 - 7$ free variables to be estimated. At the same time, Equation 6.3 introduces $n_{\mathcal{S}}$ linear equalities and θ^{C} introduces another 52. When $n_{\mathcal{P}} \geq 4$, there are more variables to estimate than there are constraints; a solution must exist. The question that remains is for what values of θ^{C}, p_{min}, and p_{max} does a *feasible solution* — i.e., one whose probabilities are all nonnegative — exist?

Similar Markov Chains have been considered in Steinsaltz and Wessel (in progress), and their work inspired some of this derivation. Essentially, they considered distributions over pitch and pair-of-interval tuples. The problem was then broken into two Linear Programming problems, whereby λ was first estimated by solving those equations that were decoupled from Λ, and then its solution was used to solve for Λ. While feasible solutions were obtained using this approach, no set of general conditions under which a feasible solution is guaranteed has been derived.

To provide insight into why a particular set of parameters can produce a set of equations for which no feasible solution exists, consider again Figure 6.7. With this graph's states, there is no way to sample a pitch whose PC lies outside of the the C to E range. Thus, an exact solution would need all of the non-obtainable PCs in θ^{P} to have zero probability. Similarly, since $\mathcal{P}$ only spans intervals between the unison and Major 3rd, larger intervals in θ^{I} would need zero probability. However, BoB's musically motivated learning priors (Section 5.7.4) ensure that no bins will ever have zero probability, so an asymptotic solution for θ^{C} cannot exist. In BoB, the issue of asymptotic convergence becomes less important because short sequences are generated.

6.3.5 External Constraints

In the exact derivation, aside from p_{min} and p_{max}, no external constraints were introduced. Nonetheless, such constraints are important because they ensure that what BoB produces can seamlessly integrate into the local environment, higher-level domain knowledge, user desires, and so on.

Prior Context Constraints

When BoB generates a real-time solo a bar at a time, the next bar used in the generation algorithm, r', must seamlessly integrate into what was most recently played in bar $\mathsf{r}' - 1$. This is accomplished by having the random walk in bar r' start in state:

$$q_{\mathsf{t}=0} = \langle p_0 = mrp_{\mathsf{r}'}, h_0 \rangle.$$

This assures that the first interval in the new walk is coupled to the MRP in the previous bar. Since there is no reason to believe that bar $\mathsf{r}' - 1$ and r' are controlled by the same DIR component, h_0 will not rely on the content in bar $\mathsf{r}' - 1$. Instead, this value will be initialized by drawing a sample from the stationary probability distribution over $\mathcal{H}$ that is implied by the goal, θ^{D}, in bar r'.

Pitch Range Constraints

One reason to constrain the possible pitch values that may be used during generation is physical — certain instruments are limited to playing in certain ranges. However, another reason to formalize range constraints is to provide an additional hook into local context, supporting, for example, the ability to let the range of activity found in a call determine the legal range of activity that is possible in a response.

Ultimately, limiting the range of possible pitches is accomplished via p_{min} and p_{max}. How these values are set for each bar depends on:

- a set of *absolute ranges*, which specifies an instrument's physical constraints;
- an incremental *relative range*, p_{incr}, which specifies how much to extend the range of pitches found in a call when producing a response.

Absolute ranges are fixed up front, never change, and place a hard limit on the maximum range of values ever allowed in $\mathcal{P}$. Relative ranges, calculated per segment, are derived for a given call.

Degeneracy Constraints

If, at a specific location within the pitch sequence, a unison interval were to occur, then its application to the VLT would produce degeneracy. A constraint that sets the smallest allowed interval, i_{min}, to either a unison or a Minor 2nd handles this issue. The VLT for whom these pitches are generated informs the algorithm what unison intervals are acceptable (Section 6.2.3). These results are stored in unison vector u:

$$u = \left\langle u_1, \cdots, u_t, \cdots, u_T \right\rangle.$$

$u_t = \texttt{true}$ indicates that the interval between ζ_{t-1} and ζ_t may be a unison.

Although degeneracy could have been addressed in Generate Stage III by cleaning up a degenerate tree after ζ was applied to it, this approach would run the risk of eliminating unison intervals that θ^{I} encouraged.

Maximum Interval Constraint

The maximum interval that may occur in a response is tied to the call in much the same way that maximum pitch range was. In particular, the maximum interval allowed in a response, i_{max}, is determined by extending the maximum interval range found in the call by i_{incr} semi-tones.

On- and Off-Beat Constraints

Some early Parker generation attempts left me with the feeling that if on-beat and off-beat pitches were handled differently, the quality of the solos might significantly improve. However, since I had already been using a stable baseline of learning results and did not want to change things mid-stream, I investigated this issue using a rather quick hack, wherein beats were identified and handled post learning. Although this attempt did not produce noticeable improved or worsened behavior, there are compelling musical reasons to provide a pitch sequence generator with explicit on- and off-beat knowledge. This quick hack is derived because it introduces the notation needed to handle beats. This derivation also demonstrates how easy it is to integrate musical common-sense knowledge into the generation algorithm.

All leftmost interior fringe children are assumed to be on-the-beat so that the VLT can calculate on-beat locations. These results are stored in on-beat vector b:

$$b = \left\langle b_{\mathrm{1}}, \cdots, b_{\mathrm{t}}, \cdots, b_{\mathrm{T}} \right\rangle.$$

$b_{\mathrm{t}} = \texttt{true}$ indicates that ζ_{t} is an on-beat. This beat labeling scheme is useful in situations like v (Figure 6.8, left), which occur when Parker plays a "run of eighths," a context in which he definitely treats the first tone in each eighth note pair differently. In contrast, v' (Figure 6.8, right) shows where this overly simple beat labeling rule breaks down. In many contexts, musicians would identify the last half-note as on-the-beat.

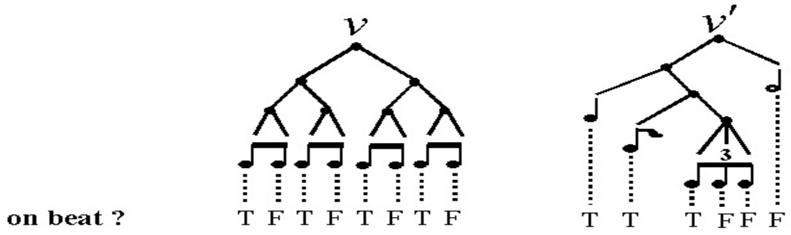


Figure 6.8: On-Beat Examples

At the very least, beat handling requires having different PC parameters for on- and off-beat events. I estimated these parameters as follows:

1. Map X^{P} into Y^{P} using learned estimate Ω^{P}.

2. For each partition of X^{P}, determine the frequency in which on- and off-beat pitch classes occur.

3. Estimate a new on- and off-beat distribution for each partition using both these frequencies and a Lagrangian prior.[5]

[5]Since all data is labeled, this solution has a closed-form.

The biggest distinctions between on- and off-beat distributions occurred with Parker's data, which makes sense given his tendency to play more chromatically than Grappelli.

Clamping Constraints

Ideally, BoB would provide a general framework for specifying arbitrary aspects of a solo's content, allowing things like the environment, common-sense musical knowledge, and specific desires that the user may wish to articulate in advance to be accommodated.

For example, there are benefits to integrating harmonic knowledge into the system, although, as I have already argued (Section 2.4.3), there are reasons why an IMC should not *need* to rely on such knowledge. One way to include this knowledge is through a constraint that requires every on-beat tone to be consonant with the underlying harmony. As another example, rhythm and pitch could be coupled by insisting that, whenever a triplet is played, the melody pivots around the middle note, changing direction by a small pair of intervals.[6]

What is needed is a mechanism for specifying that only certain pitches, intervals, and/or direction trends are allowed at given locations within a walk. In other words, at certain points BoB should be able to *clamp* particular events to one of a specific set values. For simplicity, only pitch-class clamping will be notated and derived. In particular, I will derive a method for generating a melody whose pitch at step $\mathtt{t} = c$ is guaranteed to come from a particular *clamping set*, $\mathcal{C}$, i.e., $\zeta_{\mathtt{t}=c} \in \mathcal{C}$. From this derivation, more elaborate PC, INT, and DIR extensions are straightforward.

6.3.6 The Practical Solution

Rather than solving for asymptotic convergence, BoB assumes that the individual components in $\theta^{\mathrm{C}} = \left\langle \theta^{\mathrm{P}}, \theta^{\mathrm{I}}, \theta^{\mathrm{D}} \right\rangle$ can be independently combined to produce a transition matrix that generates sequences that exhibit the desired conglomerative behavior. In total, four different transition matrices are constructed in order to handle all possible degeneracy and on-beat combinations. Using these matrices, an efficient forward-backward algorithm is derived for handling the more general clamped constraints.

[6]One listener noted that this rhythmic device, commonly used by Parker, was missing in BoB's responses.

Transition Matrices

The independence assumption allows transition matrices to be calculated in closed form. In particular:

$$\Lambda_{\mathsf{s},\mathsf{s}'} = \frac{\theta^{\mathrm{P}}_{\mathtt{m}=\mathtt{pc}(p')} \cdot \theta^{\mathrm{I}}_{\mathtt{abs}(p,p')} \cdot \theta^{\mathrm{D}}_{\mathtt{dInd}(h,\mathtt{sym}(p,p'))}}{\sum_{s'\in\mathcal{S}} \theta^{\mathrm{P}}_{\mathtt{pc}(p')} \cdot \theta^{\mathrm{I}}_{\mathtt{abs}(p,p')} \cdot \theta^{\mathrm{D}}_{\mathtt{dInd}(h,\mathtt{sym}(p,p'))}} \tag{6.10}$$

is the probability of transforming from state $\mathsf{s} = \langle p, h\rangle$ into $\mathsf{s}' = \langle p', h'\rangle$. Because of its implicit reliance on $\mathcal{P}$, this equation ensures that the pitch range constraints are met. The remaining constraints, however, still need to be provided for.

The numerator treats the probability of the next pitch as a sole function of:

- its pitch class, $\mathtt{pc}(p')$;
- the absolute interval between itself and the previous pitch, $\mathtt{abs}(p, p')$;
- the transition out of relevant directional history h via direction $\mathtt{sym}(p, p')$.

The denominator is the normalizing constant. With a sparse matrix implementation, Λ can be calculated in $\mathcal{O}(9 \cdot n_{\mathcal{P}}^2)$ time.

While there is no reason to expect that the transition matrix defined by equation 6.10 will provide long-term convergence (equations 6.6-6.9), the independence assumption will tend to produce sequences that, at least in the short term, map back into the desired goal. When a transition's conglomerative terms are, in isolation, quite likely, its unnormalized value will attenuate less than a transition whose terms do not cooperate. After normalization, those transitions that accommodate the largest conglomeration of trends will have the greatest chance of being selected. In this way, θ^{P}, θ^{I}, and θ^{D} act as a sophisticated filter, magnifying those events that are most collaborative and attenuating those that are not.

Intervallic Modification

For Equation 6.10 to be well-defined, the dimension of θ^{I} may need to be extended. When the maximum interval that is accessed by the second term, $\mathtt{abs}(p_{max}, p_{min})$, is greater than 13 (more than an octave), the dimension of θ^{I} must be increased.

When increasing θ^{I}, all intervals beyond dimension 13 are weighted according to the probability of their complement, where an *interval's complement* is defined to be the interval modulo 12 (the interval with octaves subtracted off). Intervals larger than a Major 7th are then normalized so that their total sum equals the probability originally allocated to the larger-than-an-octave bin.

θ^{I} is also modified so that the probability associated with any interval outside of the range i_{min} and i_{max} is zero.

Temporal Modification

At each step in a walk, degeneracy determines whether or not i_{min} is a unison or a Minor 2nd interval. Similarly, a different PC parameterization is used for on- and off-beats. Thus, four transition matrices are created in total, one for each possible degeneracy and beat combination. This set of matrices:

$$\mathcal{L} = \Big\langle \Lambda(\texttt{true}, \texttt{true}), \Lambda(\texttt{true}, \texttt{false}), \Lambda(\texttt{false}, \texttt{true}), \Lambda(\texttt{false}, \texttt{false}) \Big\rangle,$$

is defined by the parameters: θ^{C}, p_{min}, p_{max}, and i_{max}. When the first argument of Λ is `true`, it is calculated using the unison i_{min} interval. Similarly, when the second argument is `true`, it is calculated using the on-beat PC parameterization.

6.3.7 Forward-Backward Pitch Sequence Generation

`generatePS` (Table 6.4) takes a T step walk:

$$\zeta = \langle \zeta_1, \cdots, \zeta_t, \cdots, \zeta_{\mathsf{T}} \rangle,$$

through the chain defined by $\mathcal{L}$. Unison constraints, u, and on-beat constraints, b, are handled on a step-by-step basis, determining at each step which matrix from $\mathcal{L}$ is to be used. Prior context, which determines the walk's initial state, is handled via θ^{D} and mrp. Clamping constraints, $\mathcal{C}$ and c, are propagated through the chain's transition matrices using a variant of the forward-backwards algorithm. With a sparse matrix implementation, β, the most expensive calculation, requires in $\mathcal{O}(9 \cdot n_{\mathcal{P}}^2 \cdot c)$ floating point operations.

The walk starts in the initial state, $q_{t=0}$ (line 4 and 5), as described in the 'Prior Constraints' section. Each new state is sampled from the output transitions of the current state using the appropriate beat and degeneracy configuration (lines 6a and 9). For each execution of the `for` loop (line 6), each new state becomes the previous

$\boxed{\texttt{generatePS}(mrp, \theta^{\mathrm{D}}, \mathcal{L}, u, b, c, \mathcal{C})}$

determine the length of the walk, T, from u or b.
determine the number of states, $n_{\mathcal{S}}$, from a matrix in $\mathcal{L}$
1: make $\beta_{\mathsf{t}=c}$ a zero vector
for $p \in \mathcal{C}$, $h \in \mathcal{H}$
1a: $\beta_{\mathsf{t}=c,\mathsf{s}=<p,h>} = 1$
end
for $\mathsf{t}=c-1$; $\mathsf{t} \geq 1$; $\mathsf{t}--$
$\Lambda = \mathcal{L}(u_{\mathsf{t}+1}, b_{\mathsf{t}+1})$
2: $\beta_{\mathsf{t},\mathsf{s}} = \sum_{\mathsf{s}' \; in \; \mathcal{S}} \Lambda_{\mathsf{s},\mathsf{s}'} \cdot \beta_{\mathsf{t}+1,\mathsf{s}'}$
3: **end**
4: sample $h_{\mathsf{t}=0}$ from the stationary distribution implied by θ^{D}
5: $q_{\mathsf{t}=0} = \langle mrp, h_0 \rangle$
6: **for** $\mathsf{t}=1$; $\mathsf{t} \leq \mathsf{T}$; $\mathsf{t}++$
6a: $\Lambda = \mathcal{L}(u\mathsf{t}, b\mathsf{t})$
if $\mathsf{t} \leq c$
for $\mathsf{s}' \in \mathcal{S}$
7: $\omega_{\mathsf{s}'} = \Lambda_{\mathsf{s}=q_{\mathsf{t}-1},\mathsf{s}'} \cdot \beta_{\mathsf{t},\mathsf{s}'}$
end
normalize ω
else
8: set ω to the row in Λ corresponding to state $q_{\mathsf{t}-1}$
end
9: $q_{\mathsf{t}} = \langle p_{\mathsf{t}}, h_{\mathsf{t}} \rangle \sim \mathrm{Mnom}(1, \omega)$
10: $\zeta_{\mathsf{t}} = p_{\mathsf{t}}$
end
return $\zeta = \langle \zeta_1, \cdots, \zeta_{\mathsf{t}}, \cdots, \zeta_{\mathsf{T}} \rangle$

Table 6.4: The `generatePS` Pitch Sequence Generator

state. This process continues until all T steps have been taken. From this walk, the pitch sequence is obtained by stripping out the history part of each state tuple (line 10).

Clamping constraints are handled by c different $n_{\mathcal{S}}$-dimensional β vectors. Beta was chosen because of its use in the seminal Hidden Markov Model (HMM) paper by Rabiner and Juang (1986). The difference between their β and the one used here is that, in `generatePS`, state is not hidden. Rather than marginalizing over hidden state, β marginalizes over all possible ways of getting to a future clamped state before it is yet known how the walk will get there. The s-th element of β_{t} is defined to be:

$$\beta_{\mathsf{t},\mathsf{s}} = \Pr(q_{\mathsf{t}+1} \in \mathcal{S},\ q_{\mathsf{t}+2} \in \mathcal{S}, \cdots, q_{\mathsf{t}=c} \in \{p \in \mathcal{C};\ h \in \mathcal{H}\}, \cdots, q_{\mathsf{T}} \in \mathcal{S} | q_{\mathsf{t}} = \mathsf{s}).$$

The term of interest occurs at clamp point $q_{\mathsf{t}=c}$. In this location, only states that include a pitch from set $\mathcal{C}$ may occur.

β values are setup in lines 1 through 3. At step $\mathsf{t} = c$, β is defined so that only those states in the future that emanate out of a clamped node have a probability of occurring (line 1a). Up to and including the c-th step of the walk, vector β_t accounts for the change in transition probabilities that results when only one of a restricted subset of states may occur at step $\mathsf{t} = c$ (line 2). While $\mathsf{t} \leq c$, the probability of a next state depends on β and Λ (line 7). Once the walk has passed clamp c, there is no subset over which to marginalize, and so β is no longer used (line 8). It is only because of the chain's Markov assumption that the calculation of β is efficient, requiring a look ahead of one step (line 2).

6.3.8 Searching for a Good Pitch Sequence

Although `generatePS` ensures that the desired constraints will be met, there is no guarantee that a particular finite walk will map into the desired goal.[7] However, because in real musical contexts about 50% of the walks achieve the desired conglomerative behavior (Section 6.5), the odds of generating a sequence that maps into the goal significantly increases when the `generatePS` distribution is re-sampled several times.

`searchPS` (Table 6.5) samples `generatePS` n_{walks} times. The distribution being sampled is defined by the $mrp, \theta^{\mathrm{D}}, \mathcal{L}, u, b, c,$ and $\mathcal{C}$ arguments. `searchPS` also specifies

[7] For finite walks, this is also an issue with the exact solution.

$\boxed{\texttt{searchPS}(mrp, \theta^{\mathrm{D}}, \mathcal{L}, u, b, c, \mathcal{C}, y^{\mathrm{C}}, l^{\mathrm{C}}, \Omega^{\mathrm{C}})}$

$\quad$ $success = \texttt{false}$; $n_{goals} = 0$; $n_{subgoals} = -1$; $diff = \infty$
$\quad$ $\texttt{for}\ n = 1;\ n \leq n_{walks};\ n{+}{+}$
1: $\quad\quad$ $\zeta_{try} = \texttt{generatePS}(mrp, \theta^{\mathrm{D}}, \mathcal{L}, u, b, c, \mathcal{C})$
2: $\quad\quad$ $x^{\mathrm{C}} = \texttt{featuresCGL}(mrp, \zeta_{try})$
3: $\quad\quad$ $\langle y^{\mathrm{C}}_{try}, l^{\mathrm{C}}_{try} \rangle = \texttt{classifyCGL}(x^{\mathrm{C}}, \Omega^{\mathrm{C}})$
4: $\quad\quad$ $\texttt{if}\ y^{\mathrm{C}} == y^{\mathrm{C}}_{try}$
$\quad\quad\quad$ $success = \texttt{true}$
$\quad\quad\quad$ $n_{goals}{+}{+}$
$\quad\quad\quad$ $\texttt{if}\ |l^{\mathrm{C}} - l^{\mathrm{C}}_{try}| < diff$
5: $\quad\quad\quad\quad$ $\zeta_* = \zeta_{try}$; $diff = |l^{\mathrm{C}} - l^{\mathrm{C}}_{try}|$
$\quad\quad\quad$ $\texttt{end}$
6: $\quad\quad$ $\texttt{elseif}\ !success$
7: $\quad\quad\quad$ $matches$ = number of subgoals shared between y^{C} and y^{C}_{try}
$\quad\quad\quad$ $\texttt{if}\ n_{subgoal} < matches$
8: $\quad\quad\quad\quad$ $\zeta_* = \texttt{t}_{try}$; $n_{subgoal} = matches$
$\quad\quad\quad$ $\texttt{end}$
$\quad\quad$ $\texttt{end}$
$\quad$ $\texttt{end}$
$\quad$ $rate = \frac{n_{goal}}{n_{walks}}$
$\quad$ $\texttt{return}\ \langle \zeta_*, success, rate \rangle$

Table 6.5: The `searchPS` Pitch Sequence Generator

how to determine which of these walks, ζ_*, is "best" — a concept that is defined by the y^{C}, l^{C}, and Ω^{C} arguments. Two different user models are presented (once via $\mathcal{L}$ and again via Ω^{C}) so that different models can be used to drive generation and assess fitness. The former model is called the *generative model* and the latter is called the *search control model*.

When at least one walk maps into the desired goal, `searchPS` is considered *successful* (the *success* value that is returned is set to `true`). Otherwise, `searchPS` is considered a *failure*. When `searchPS` succeeds, how *easy* its generation task was is quantified by *rate*, which records how many of the individual walks that were sampled mapped into the desired goal.

Each execution of the outermost loop samples a new walk, ζ_{try} (line 1). This walk must be classified before its quality can be assessed (lines 2 and 3). When defining a "best" walk, two special cases must be handed:

1. No walk may map into the goal, in which case the "next best thing" needs to be

defined.

2. When more than one walk maps into the desired class, an additional distinction for determining "best" is needed.

Initially, assessment is concerned with finding any walk that maps into goal y^{C} (the equality in line 4). Until a walk that maps into y^{C} is found, the "next best thing" is defined by how many of the underlying *subgoals* it achieves (lines 7 and 8). Each individual component in y^{C} is treated as a different subgoal, so when all three subgoals match, the control loop at line 4 (rather than line 6) is executed. Once a goal sequence has been found, only walks that achieve this goal with a more similar surprise (i.e., labeled likelihood, Section 5.10.5) are considered better (line 5).

6.4 Complete vMn-Based Generation Algorithm

Algorithm `vMnGenerate` (Table 6.6) details how Generate Stages I, II, and III are combined into the complete algorithm. For convenience, generation is only passed the bar numbers for call and response, r and r', because these can be used to access those variables that change per segment (i.e., lines 1, 5, 10, and 12). All other generation-related parameters — rhythmic control (γ, γ', γ''), absolute range constraints, relative range constraints (p_{incr}, i_{incr}), the number of walks (n_{seeds}), and the generative and search control models — are obtained as part of a particular experiment's *generation setup*. After generating a response, `vMnGenerate` automatically incorporates its result into the environment and the agent's state using index r' (line 12).

The values of a particular generation setup are fixed up-front, before online interaction begins. Absolute pitch range values are obtained by examining the training set for its minimum and maximum pitches values. For Parker's training set, the E♭ : 3 to A : 5 range is obtained; for Grappelli, the range is G : 3 to A : 6.

Within `vMnGenerate`, when call and response variables must be distinguished, primes are used. The call's ranges are determined before its rhythm is tweaked (line 2). These ranges, the absolute and relative pitch range constraints, and the maximum interval range constraint, combine to specify what values of p_{min}, p_{max}, and i_{max} to use when calculating $\mathcal{L}$. Next, rhythm tweaking transforms the call's VLT (line 3), and from this result, degeneracy constraints are determined (line 4).

Another generation setup parameter, *last*, indicates when *last clamping* is active

$\boxed{\texttt{vMnGenerate}(\mathsf{r}, \mathsf{r}', \})}$

1: retrieve v, y^{C}, and l^{C} for call bar r

2: determine p_{min}, p_{max}, and i_{max} using both the ranges found in v
  and the range constraint constants

3: $v' = v.\texttt{recursiveTweak}()$

4: calculate u and b for v'

5: retrieve mrp' for response bar r'

6: calculate $\mathcal{L}$ using y^{C}, p_{min}, p_{max}, i_{max}, and the generative user model
  retrieve the search control model, $\Omega^{\mathrm{C}}_{search}$

8: `if` *last*
  determine length of the walk, T', from u or b

9: $c = T'$

10: $\mathcal{C} = \{mrp_{\mathsf{r}+1}\}$
  `end`

11: $\langle \zeta, success, rate \rangle = \texttt{searchPS}(mrp', \theta^{\mathrm{D}}, \mathcal{L}, u, b, c, \mathcal{C}, y^{\mathrm{C}}, l^{\mathrm{C}}, \Omega^{\mathrm{C}}_{search})$

11a: assign ζ to non-rest leaves of v'

12: update the environment concerning v' and $mrp_{\mathsf{r}'+1}$

Table 6.6: The Complete `vMnGenerate` Generation Algorithm

(line 8). Last clamping forces the last pitch in the response (line 9) to be identical to the last pitch in the musican's call (line 10). In this situation, goal-driven generation is essentially being asked to "fill in the blanks" between two pitches using a specific conglomerative trend. The pitches to assign to v' are then sampled and assigned to v' (lines 11 and 11a) The final task is informing the environment about the new response (line 12). This task includes scheduling v' into a real-time buffer for playback in bar r' and, given this playback, updating the MRP.

`vMnGenerate` only tweaks rhythm once (line 3), but samples n_{walks} different pitch sequences (line 11). This decision was made because v' affects u, b, and c, which directly impact the most expensive calculation, β. While β is expensive, its amortization over n_{walks} makes it real-time feasible. For example, in the current implementation, approximately 2.5 seconds are needed to calculate β and $\mathcal{L}$, with which a walk takes about 1 [ms] to sample. For live performance, an execution time on the order of 1.5 seconds per segment is needed. Fortunately, it should be easy to reduce the cost of β because no optimizations (sparse matrix calculations, etc.) have yet been tried.

6.4.1 Simulated Music Datasets

A simulated dataset for a musician's training data is created with the following steps:

1. Retrieve the entire set of musician's improvisation segments (including the pickup bar).
2. Strip out any segments that contain only rests (no pitches will be generated for these data-points; they are uninteresting).
3. Generate one long response for the call that is comprised of what is left in this sequence of segments.
4. Use this response improvisation to construct a new conglomerative histogram dataset.

Because this procedure generates very long solos, wandering becomes a significant issue. *Wandering* occurs because, over the long term, the previous MRP controls where in $\mathcal{P}$ the next segment is likely to go. As generation progresses from segment to segment, there is more and more opportunity for the solo's pitch content to gravitate towards either end of $\mathcal{P}$'s range. Last clamping is very effective in this circumstance because it forces the simulated solo response to have the same MRPs that were found in the musician's warmup data. Thus, data simulated with last clamping is more tightly coupled to the training data. When clamping is inactive, only the relative range constraints restrict wandering (by modifying the range of $\mathcal{P}$ on a per-segment basis). Typically, without last clamping, incremental ranges must be significantly smaller (e.g., Table 6.7) before results of similar musical quality are produced.

6.4.2 Solo Trading

When trading solos, the cut-and-paste scenario (Section 3.3.3) is used, so the call's tonal abstractions may not be compatible with the underlying harmony that accompanies the response. Hence, cut-and-paste is modified as follows. Given that: 1) bar $\mathtt{r}$ is to be generated, 2) the underlying harmony contains n_{harm} bars per chorus, and 3) the training set's first chorus bar is number n_{offset}, the PC goal is copied from one of the training bars in column:

$$\mathtt{mod}(\mathtt{r} - n_{offset}, n_{harm}).$$

For example, the columns in Figure 3.1 correspond to $n_{harm} = 12$ and $n_{offset} = 2$.

Cut and paste for the INT and DIR goals proceeds without modification:

$$y^{\mathrm{I}}_{\mathsf{r}} = y^{\mathrm{I}}_{\mathsf{r}-length};\qquad (6.11)$$
$$y^{\mathrm{D}}_{\mathsf{r}} = y^{\mathrm{D}}_{\mathsf{r}-length}.$$

length was defined on page 62.

6.4.3 Other Interaction Scenarios

Only the simple abstract cut-and-paste interaction has currently been considered, but there is nothing inherent in `vMnGenerate`'s strategy that keeps it from handling richer interaction scenarios.

One intriguing and powerful possibility that others have also argued for involves hierarchical, multi-resolution control of musical sequences (Mozer 1994; Höthker 1999; Hörnel 1998). For example, imagine having BoB use a call's entire abstract sequence to predict (rather than copy) what goal sequence is used to control response. All that would be needed to support this richer scheme is a higher-level sequence predictor. Similar prediction techniques could also be applied to vary aspects of generation that are currently treated as constants. For example, how much rhythm is embellished or simplified from bar to bar could be estimated on a per-segment basis.

More general interaction scenarios are also possible. Clamping provides a plethora of opportunities, allowing the user to customize their own environment, introduce domain knowledge, etc. For example, the lead-sheet abstraction could be extended to allow specific events (harmonies, scale preferences, contours, rhythmic transformation properties) to be "snapped in" at specific time points. The importance of this type of user control has been demonstrated elsewhere (Abrams, Oppenheim, Pazel, and Wright 1999), and its integration into BoB's generation algorithm would be relatively straightforward.

6.5 Performance

The practical conglomerative-driven generation approach is empirically validated in two ways. Both evaluations use simulated music data, so performance is measured in real music contexts. First, how much better the practical generation scheme is than random guessing is investigated. Next, the degree to which this generation scheme reproduces the underlying user model's structure is evaluated.

6.5.1 Compared to Random Guessing

One way to evaluate how well the practical independence assumption works is to estimate how much it outperforms a method that knows nothing about the learned user model. This latter method is implemented as a completely non-informative vMn model, where all component values are identical and uniformly distributed.

This comparison is measured by simulating two different datasets and contrasting the performance of their `searchPS` algorithms. In both simulations, the search control model that defines performance is one of the learned user models described in Table 5.8. In one dataset, the generative model is non-informative; in the other dataset, generation is based on the learned model. In total, eight datasets (Table 6.7) were simulated. Each dataset is referred to by a scenario index, which identifies its generation setup.

This set of experiments focuses on measuring how well conglomerative goals are achieved. (In order to control the experiments as much as possible, rhythms were not modified.) Although it has been my experience that goals are usually met with a smaller search (e.g., $n_{walks} = 25$), the value of 150 was used to provide more accurate *rate* estimates.

setup: $\gamma = \gamma' = 0$, $\gamma'' = 1$, $n_{walks} = 150$

Simulated Music Datasets			Generative Setup			
Scenario	Musician	Number of Bars	Generative Model	Last Clamping	p_{incr}	i_{incr}
I	Parker	120	uniform	inactive	3	3
II	Parker	120	learned	inactive	3	3
III	Parker	120	uniform	active	12	12
IV	Parker	120	learned	active	12	12
V	Grappelli	129	uniform	inactive	3	3
VI	Grappelli	129	learned	inactive	3	3
VII	Grappelli	129	uniform	active	12	12
VIII	Grappelli	129	learned	active	12	12

Table 6.7: Simulated Music Data

The performance of `searchPS` scenario is summarized in Table 6.8. The Success Rate statistics that are reported were only compiled for those bars that were successfully generated. These rates reflect how easy it was to simulate the successful data's bars. This quantification is important because it impacts how large n_{walks} must be to succeed most of the time. The histograms in Figure 6.9 provide more detail about the success rate distribution for scenarios I and II.

Performance Results						
	Number of	Number of	Success Rate			
Scenario	Success Bars	Failure Bars	Ave	Std	Min	Max
I	92	28	0.037	0.045	0.0067	0.33
II	120	0	0.4	0.19	0.033	0.81
III	98	22	0.043	0.1	0.0067	1.0
IV	120	0	0.47	0.21	0.033	1.0
V	66	63	0.052	0.089	0.0067	0.51
VI	122	7	0.45	0.24	0.0067	0.94
VII	59	70	0.075	0.19	0.0067	1.0
VIII	125	4	0.56	0.24	0.0067	1.0

Table 6.8: `SearchPS` Performance

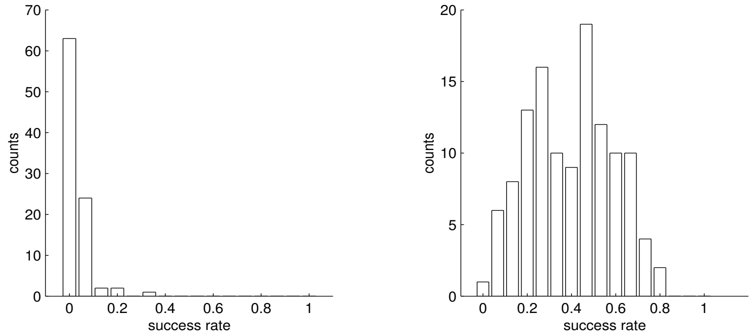


Figure 6.9: Ease in Generation for Scenario I (left) and II (right)

Clearly, the non-informed generator accomplishes many fewer goals than the corresponding learned generator. In the worst case (VII), 70 out of 129 goals fail. In other words, 70 of the simulated bars did not exhibit the playing modes that were assigned to the original Grappelli data. Learning, on the other hand, is quite successful. All of the simulated Parker bars succeed, and only a few of the Grappelli bars fail.[8] Not

[8]For those learn-based bars that did fail, only two of them did so because a sequence with the correct PC class was never found. In all other cases, the DIR mapping failed.

only is the learned generator more successful, its success rates are much larger, which means that more of the algorithm's time can be spent fine-tuning a bar's surprise (as opposed to searching for a sequence that maps into the right playing mode).

In all but the non-informed Grappelli case, clamping improves performance. The most likely explanation is that, as a side-effect of more tightly coupling the simulation task to the original dataset, the space of possible searches becomes more highly focused in a useful way. For example, in the most extreme case (2 of Grappelli's bars, and one of Parker's) clamping assured success because it fixed the bar's only pitch value (hence the 1.0 Max Success Rates).

6.5.2 Compared to the Original Training Data

Another useful indicator of how well the practical independence assumption works is obtained by comparing the trends found in the original dataset to the trends exhibited in the simulated data. In Figure 6.10, comparisons between the distribution of Parker's original and simulated PC, INT, and DIR counts are presented. The dotted bars reflect the distributions found in Parker's training data; the solid bars correspond to a simulated dataset's distribution. For simulation scenarios, II and IV, refer to Table 6.7. Histogram bin labels were described in Sections 4.4.3, 4.4.4, and 4.4.6.

The important point in each of these figures is that a distribution corresponds to a dataset's overall trend. For example, the solid bars in plot (a) show how all the counts in Parker's PC training data are distributed when all the bars are viewed together. Similarly, the dotted bars in plot (a) display how the counts produced by simulation scenario II are distributed. Although there is not a big difference between the clamped and un-clamped figures, the difference between the original and simulated distributions is somewhat larger in the un-clamped scenario, especially with the PC distribution.

6.5.3 With Respect to the Underlying Model's Structure

While it is encouraging that conglomerative generation successfully responds to all (most) of Parker's (Grappelli's) calls, an even stronger indication of how well the practical approach works can be obtained by running further learning experiments on these simulated datasets. In the experiments presented here, how well the practical

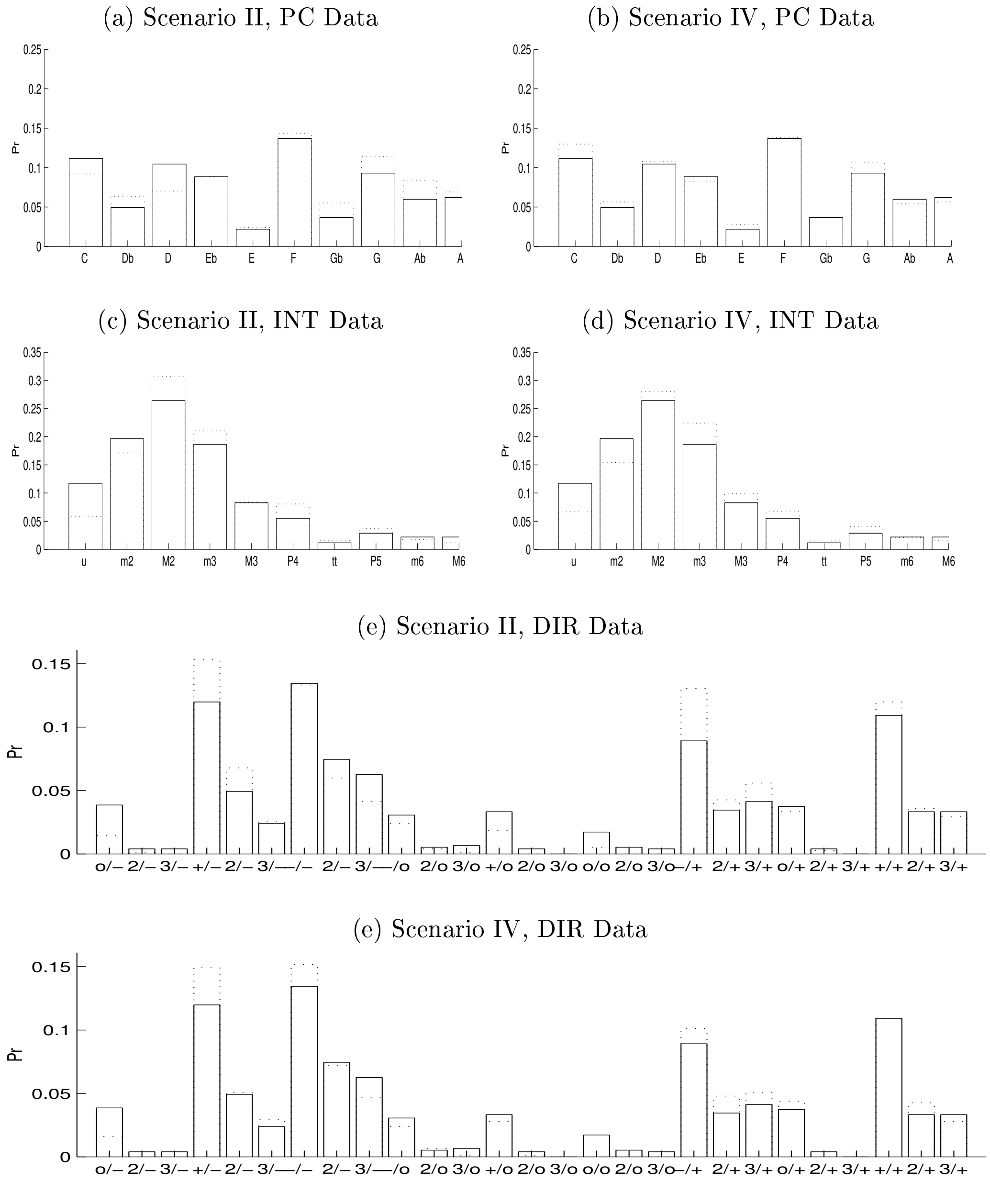


Figure 6.10: Distribution of Counts in Parker's Training and Simulated Data

generation scheme closes the loop — integrating learned knowledge into the generation process — is measured by comparing how similar a model learned for the simulated data is to the original learned model.

This evaluation involves the following steps:

1. Learn $\Omega^{\boldsymbol{Tr},\mathrm{C}}$ and $Y^{\boldsymbol{Tr},\mathrm{C}}$ from $X^{\boldsymbol{Tr},\mathrm{C}}$ (the learned models reported in Table 5.8 are used).

2. Generate simulated dataset X^{C} using $\Omega^{\boldsymbol{Tr},\mathrm{C}}$ for both the search control and generation models (simulations from Table 6.7 are used).

3. Learn Ω^{C} and Y^{C} from X^{C}.

4. Obtain an additional partition for each dataset using the other dataset's classifier:

$$\begin{aligned} {Y'}^{\boldsymbol{Tr},\mathrm{C}} &= \texttt{classifyVmn}(X^{\boldsymbol{Tr},\mathrm{C}}, \Omega^{\mathrm{C}}); \\ {Y'}^{\mathrm{C}} &= \texttt{classifyVmn}(X^{\mathrm{C}}, \Omega^{\boldsymbol{Tr},\mathrm{C}}). \end{aligned}$$

5. Estimate the disagreement between the learned models by determining the average difference between their partitions:

$$\begin{aligned} dis^{\mathrm{P}} &= \frac{\texttt{diff}(Y^{\boldsymbol{Tr},\mathrm{P}}, {Y'}^{\boldsymbol{Tr},\mathrm{P}}, \texttt{true}) + \texttt{diff}(\mathtt{Y}^{\mathrm{P}}, \mathtt{Y'}^{\mathrm{P}}, \texttt{true})}{2}, \\ dis^{\mathrm{I}} &= \frac{\texttt{diff}(Y^{\boldsymbol{Tr},\mathrm{I}}, {Y'}^{\boldsymbol{Tr},\mathrm{I}}, \texttt{true}) + \texttt{diff}(\mathtt{Y}^{\mathrm{I}}, \mathtt{Y'}^{\mathrm{I}}, \texttt{true})}{2}, \\ dis^{\mathrm{D}} &= \frac{\texttt{diff}(Y^{\boldsymbol{Tr},\mathrm{D}}, {Y'}^{\boldsymbol{Tr},\mathrm{D}}, \texttt{true}) + \texttt{diff}(\mathtt{Y}^{\mathrm{D}}, \mathtt{Y'}^{\mathrm{D}}, \texttt{true})}{2}. \end{aligned}$$

   `diff` is defined in Appendix D.

The larger these disagreements are, the less compatible the two learned model's underlying structures are likely to be. While some of this disagreement can be explained in terms of "learning noise," i.e., the original model's expected total error rate (*hardness* plus *cost*), some of it can also be attributed to generation not exactly replicating the vMn model's structure.

PC, INT, and DIR model discrepancies were estimated for simulation scenarios II, IV, VI, and VIII. The results are presented in Table 6.9. For comparison, the corresponding expected total error rate (Column 'Tot') is displayed (this rate comes from Table 5.10). To get a feel for what these numbers mean, consider the worst-case possibility: random guessing (Column 'Rand').

Datasets		PC Model			INT Model			DIR Model		
Musician	Simulation	dis^{P}	Tot	Rand	dis^{I}	Tot	Rand	dis^{D}	Tot	Rand
Parker	II IV	0.45 0.16	.32	.80	0.25 0.35	.31	.75	0.05 0.11	.09	.66
Grappelli	VI VIII	0.09 0.08	.18	.75	0.09 0.13	.23	.80	0.12 0.29	.07	.75

Table 6.9: Relearn Generation Experiments

When compared to random performance, there is no denying that generation can reproduce a fair amount of the learned vMn model's structure. Overall, the lowest discrepancies were obtained with Grappelli's un-clamped simulation (scenario IV). In this case, PC and INT discrepancies are half of what can be explained by "learning noise" alone. The DIR discrepancy, on the other hand, is twice as large as its "learning noise." Nonetheless, these numbers indicate that most of the underlying structure that was learned to explain Grappelli's training data is in fact reproduced by the generation algorithm. For the clamped Grappelli case, PC discrepancy decreases ever so slightly (as it should given the additional coupling it provides) at the expense of INT and DIR performance. With Parker, clamping greatly improves PC's discrepancy (reducing it from 45 to 16%), but once again this comes at the expense of INT and DIR structural loss. That clamping influences Parker's PC discrepancy so much is likely due him playing more chromatically than Grappelli.

6.6 Discussion

6.6.1 Rhythm and Pitch

In the music perception community, there is open debate concerning whether or not the treatment of rhythm and pitch should be unified or whether they can be treated as independent dimensions (Boltz 1999). In BoB, the coupling between rhythm and pitch content is relatively weak — the number of leaves and the presence of syncopation being the only direct links. Fortunately, stronger ties with rhythm can be supported with clamping constraints.

Handling other important aspects of rhythm will require extending the CGL feature space. For example, rests are currently ignored, even though these are perceptually important (serving to mark various parts of a motive or phrase, etc). In addition, no distinction is made between compound and composite durations. Rather, both durations produce additional PC counts, and both produce unison interval and zero-motion related counts even though musically, these correspond to different phenomena (syncopation and note repetition respectively).

When I was first creating the system and analyzed Parker's data, this distinction seemed unimportant (only 1.6% of his 870 PC counts were attributed to note repetition, as were 13% of his 102 unison interval counts). Later on, when I looked at Grappelli's data, repetition played a more serious role (causing 4.2% of his 867 PC counts and 24% of his 149 unison intervals). In hindsight, it is easy to see that this issue deserves more explicit consideration and should be dealt with in future work.

6.6.2 Coupling Learning and Generation

For goals to be achieved in real-time, a pitch-sequence sampling procedure that tightly couples with Ω^{C} is needed. While many music generation schemes, e.g., BoB, *EMI* (Cope 1992), *CHIME* (Franklin 2001), *GenJam* (Biles 1994), are cast in a *generate-and-test* framework, the explicit and principled coupling between `generatePS` and θ^{C} distinguishes this algorithm from other approaches. In particular, with an exact solution, multiple samples would only be needed because a given sample might fall into a competing component's decision boundary. Although with the practical approach no such guarantee exists, in real musical contexts, a single goal-based generation attempt can be expected to succeed on the order of 50% of the time.[9] In other words, after a reasonably small number of `searchPS` trials, failure becomes unlikely. For example, with n_{walks} =10, on average, about 0.1% can be expected to fail.

6.6.3 Conglomerative Goals and Constraints

The practical sampling scheme provides principled methods for incorporating constraints directly into a playing mode driven Markov chain. These constraints are serviced in two ways. First, multiple transition matrices are provided, which allows various range, degeneracy and beat-related constraints to be realized. Second, a forward-backwards propagation algorithm is provided, which allows a fairly general set of clamping constraints to be met.

[9]This estimate is based on the average rate values obtained for the learned generative models in Table 6.8.

I cannot over-emphasize the computational power that this goal-driven, constraint-ensuring combination provides. For example, early in my research, I built a probabilistic solo-modifying algorithm that tweaked both pitch and rhythm simultaneously. Due to the hierarchical nature of the rhythmic encoding (VLTs were used), the resulting rhythms were usually acceptable. The pitches that were assigned to these rhythms, however, often strongly violated one's musical common sense. I had originally been motivated by the idea of "tweaking a particular branch in a VLT, while ensuring that it realized a particular playing mode", but I lacked a method for generating a pitch sequence that, on the one hand, realized this mode, and on the other hand, started and ended on predefined pitches. What I really wanted was an algorithm that could perform both *melody completion*[10] and run in real time. `vMnGenerate` fulfills this need.

While there are ample occasions where the solos BoB generates using `vMnGenerate` violate one's musical common sense, the clamping constraints provide a mechanism for eliminating some of this egregious behavior. For example, one listener noted that, in particular solos, a scale's Minor 3rd tone should not have been played because, in its evolving context, its Major 3rd had just been played. Violations of this sort could be handled by a particular choice of interval clamping constraints (and such reasoning is only possible because the model explicitly encodes intervals into its representation). **That such violations occur merely indicate the obvious: although CGL features capture important aspects of a melody's temporal content, they do not capture all salient aspects. In this light, constraints can be viewed as providing a "back-door" mechanism for manually imposing additional temporal structure onto a particular mode's generator.**

This observation naturally leads one to consider the possibility of learning these additional constraints directly from music data. Whether or not such a concept can be learned is an open research question, but I am optimistic. Recall that sample size sparsity is a mechanism by which specific histograms can reveal partial information about their underlying component generators (Section 5.10.3). Provided the vMn model captures real playing mode distinctions, it is reasonable to consider inferring which parts of these components are revealed in certain contexts.

While constraints are valuable, they can also decrease the strength of coupling between θ^{C} and the `generatePS` pitch sequence distribution. For example, this is

[10] As was done in the work of Bellgard (1993)

exactly the affect that the β vectors have. It is also easy to dream up situations in which realistic constraints would make goal-based generation difficult. For example, suppose BoB was generating a pitch sequence of length four, and this was to be assigned to the leaves of a VLT that contained four quarter-notes. Should θ^{C} expect unison intervals 90% of the time, degeneracy would allow at most 50% of the tree's intervals to be unisons. Clamping the last note in a sequence presents a similar conflict when the MRP is significantly lower (or higher) than the clamp's pitch and θ^{D} strongly prefers downwards (or upwards) motions.

Fortunately, preliminary simulations seem to indicate that constraints and goals can be successfully realized in real musical situations (Section 6.5). However, further research needs to be done. For instance, it is worth pursuing a generation method that relies on a parameterization that has been conditioned so as to optimize goal y^{C} for a particular set of constraints.

6.6.4 Convergence and Sparsity

Given how sample size impacts `classifyVmn`'s discrimination power, one might think that walks that are shorter would be more difficult to successfully generate. On the other hand, since the conglomerative goal and additional constraints together impose a set of temporal constraints, one might expect short walks to be easier to generate because their search spaces are smaller.

One reason this generation scheme works in practice is because of how sparse per-bar histograms are. This sparsity can be thought of in terms of providing enough "noise" to allow the sampling algorithm to "adequately" explore its solo space.[11] In particular, with fewer counts, there are more opportunities for a given sample's basic histogram shape to vary.

That walks are short also hides that the sampling scheme is inexact and more counts will only clarify any divergence that may exist between the `generatePS` distribution and θ^{C}. Divergence alone, however, does not determine a sampling scheme's success rate. Rather, success depends upon how much a given component's `generatePS` distribution falls within the corresponding goal's decision boundary. Typically, it is easier to learn boundaries than it is to learn entire distributions, so measuring per-

[11] Since this exploration heavily prefers collaborative PC, INT, and DIR combinations, this "noise" is hardly random.

formance in terms of these boundaries also contributes to the generation algorithm's success.

Chapter 7

Musical Evaluation

In this chapter, the performance of BoB's user-specific perception and generation components are assessed from a musical point of view. The former is assessed from a qualitative, musical basis, and the latter is assessed in terms of empirical human listening experiments. Parametric details concerning the learned Parker and Grappelli models, the musical appropriateness embedded within these details, and the musically reasonable ways in which these details operate upon the training data are first presented. Given BoB's unsupervised learning paradigm, this validation is especially important because it provides an assurance that, although various vMn modeling assumptions are violated when a musician generates their solos, in practice, useful and salient musical structure is learned. Next, the degree to which these two learned models and their corresponding components produce behaviors that are noticeable to human listeners is discussed. This chapter is primarily written for a musically familiar audience; a basic understanding of harmonies, scales, and other basic musical concepts is assumed.

7.1 Overview

With respect to user-specific perception, Parker's learned PC model is the focus of attention because, through this analysis, the basic methodology for reasoning about the other models follows. In addition, this analysis powerfully demonstrates two key benefits of applying a vMn model to short melodic segments. These benefits arise because the components themselves, as well as their use in perceiving a specific, temporally ordered training set, provide a very flexible and distributed representation that, musically speaking, happens to capture sensible, musician-specific information.

The analysis of Parker's INT and DIR models are geared towards shedding light on the musical meaning that these models' various feature combinations capture. With the additional data provided in Appendix E, readers can perform more detailed analyses for these models if desired. The same thing is true of Grappelli's model, which is detailed in Appendix F. To conclude the musical analysis of user-specific perception, several sets of examples from Parker's and Grappelli's training data, each set mapping into a different CGL playing mode, are presented, contrasted, and compared to provide a more concrete idea of how the vMn perception mechanism works when perceiving short melodic segments.

With respect to generation, two different types of listening experiments will be outlined, each exploring a specific question. In the *playing mode variation* experiments, subjects were asked to listen to pairs of segments and assess whether or not they thought the two were generated by the same or different playing modes. The hypothesis in this experiment was that listeners would be able to tell the difference, and, to the extent that they could, this would justify the need for using a mixture model that can provide multiple components. In the *learn model variation* experiments, subjects were asked to first listen to a call from a particular musician's training set, then listen to a pair of responses to this call, and assess whether or not each of these responses was generated using the same or different learned models. Once again, the hypothesis in this experiment was that listeners would be able to tell the difference, and, to the extent that they could, this would justify the need for user-specific learning.

7.2 Customized Perception: Charlie Parker

In this section, the user model learned for Charlie Parker's training data, $X^{\boldsymbol{P},\mathrm{C}}$, is described. The score used to construct this training set is in Appendix E.1. This model is the same one that was first presented from a machine learning point of view in Tables 5.8 and 5.10.

7.2.1 Learned Tonality

The five pitch-class components that were learned for Parker, i.e., $\theta^{\boldsymbol{P},\mathrm{P}}_{\mathsf{k}=1}, \theta^{\boldsymbol{P},\mathrm{P}}_{\mathsf{k}=2}, \cdots, \theta^{\boldsymbol{P},\mathrm{P}}_{\mathsf{k}=5}$, are shown in the first five rows of Table 7.1. Both a symbol and a number (far left columns) refer to the same component, e.g., ‘ ’ and ‘1’ both refer to the component in row one. The former symbol is used in figures, e.g., Figure 7.1, and the latter is used in scores, e.g., Table 7.4.

Since each PC component identifies a preference for certain pitch classes, it makes sense to consider how these components lend themselves to musical scale-based interpretations. For each component, strong correlations between their more probable bins and a particular scale's pitch-class set can be found. This correspondence led me to interpret each as identified in the far right column (the background gray columns identify this scale's pitch-class set). With components , , and , these interpretations are immediately obvious: a one-to-one correspondence between a scale's pitch-class set and a component's more probable bins exists. Regardless of scalar interpretation, however, all that BoB knows is that each component has a certain mixture weight and pitch-class distribution.

The ‘Size’ column contains two equivalent numbers, how many bars in $X^{\boldsymbol{P},\mathrm{P}}$ were assigned to a component, and what percentage of the training set ($\mathsf{N}^{\boldsymbol{P}} = 119$) this number corresponds to. A component's size is related to its mixture weight, π, since this value is the posterior-weighted sum of the training data (Equation 5.12). How this model partitioned each of Parker's *Mohawk* bars is presented in Appendix E.2.1 through E.2.5.

The pitch-class distribution in the bottom row, the *single mixture*, presents how BoB would have modeled Parker had it been forced to fit a single Multinomial component to his entire dataset. This model reflects the distribution of pitch classes per improvisation, while the $\mathsf{k} = 5$ model demonstrates how the tonality in different bars can be partitioned into distinct regions. That multiple component models estimate specific pitch-class distributions using averages constructed primarily from “like-minded” histograms is crucial — only the most mundane musician would play a solo whose pitch-class distribution was constant throughout. Similarly, there is no reason to expect that a single tonal distribution would adequately describe a musician's behavior in a fixed harmonic region.[1] It is useful to compare these models by

[1]Which was what was assumed in the Dannenberg and Mont-Reynaud (1987) model. In particular, a single component for each eighth-note in a blues progression was used, which amounts to

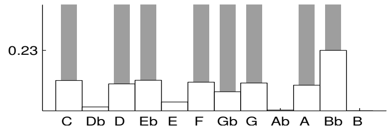
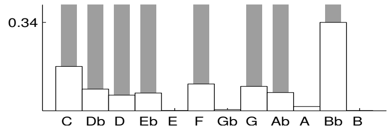
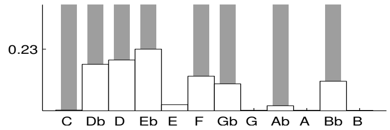
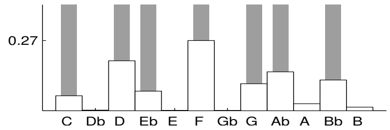
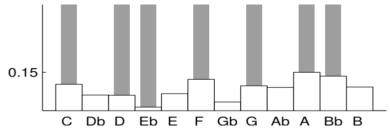
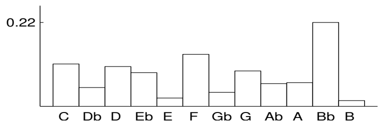
Symbol		Pitch-Class Probabilities	Size		π	Associated Scale
	1		34	28.6%	0.309	B♭ Major Bebop scale
	2		41	34.5%	.310	E♭ Bebop-7 scale
	3		10	8.4%	.079	E♭ Min+7 scale
	4		23	19.3%	.186	E♭ Major scale
	5		11	9.2%	.115	B♭ Major scale (with chromaticism)
			a single mixture (for comparison)			

Table 7.1: Parker's Learned PC Components

considering which parts of the single mixture's probabilities split into which parts of the vMn's five components; this comparison highlights exactly how the mixture model enlarges BoB's perception of Parker's tonal usage. Inventive musicians will use different scales (or at least different realizations of the same scale) in different choruses.

The first two components, which account for a substantial part of the training set (63%), have straightforward Bebop scalar interpretations. **These interpretations lend impressive credibility to the algorithm's ability to learn musician-specific results because the Bebop scales were defined in part so as to explain the improvisational style that Parker had a hand in inventing** (Ostransky 1977; Baker 1983). The harmonic context of *Mohawk* lends further credibility to the musical appropriateness of what was learned. For a B♭ blues progression, it makes perfect sense to base one's improvisations on B♭ and E♭ scales.

A closer look at Table 7.1 sheds light on what I mean when I say that this model is a *distributed representation*.[2] For example, consider the component. Rather than viewing this component as an abstract, symbolic B♭ Major Bebop scale, BoB sees it as something that prefers pitch class B♭ about twice as much as its other probable pitches (C, D, E♭, F, G♭, G and A♭). However, when perceiving a histogram that contains twice as many B♭ counts as it does C counts, both components and will appear to be likely generators. In this case, an absence of D♭ counts becomes a crucial indicator, as does the presence of A counts. In the vMn model, I use the term 'distributedness' to refer to the fact that various component's features collaborate and/or inhibit one another during perception — the overall result is somewhat emergent. In the context of generation, this distributed representation is all the more useful than a symbolic, scalar view because it provides information about *how* the user realizes particular pitch sequences.

Spontaneously creating melody, an act as simple as improvised humming, is as much about one's ingenious use of exceptions as it is about one's adhering to certain rules (Loy 1991; Papadopoulos and Wiggins 1999; Johnson-Laird 1991). Since the dataset determines how probable a component's pitch classes are, BoB is free to construct a palette that accommodates both an artist's choice of scales and their subsequent, personalized modification. For example, a reasonable interpretation of

breaking Figure 3.1 into 96 columns and fitting a single PC probability distribution to each column.

[2]While mixture models can be implemented in terms of neural networks, my use of the term 'distributed' is higher-level than what is typically referred to when discussing neural nets.

the ■ component is the following synthesis of musical ideas:

> Improvise on the B♭ Major scale, but prefer the tritone (E) over the scale's 4th (E♭), and add blues chromatic tones (the Minor 2nd, 3rd, and 7th) to the mix.

While such concepts are roughly described in jazz texts, e.g., (Baker 1983; Russell 1959; Coker 1970; Ostransky 1977; Aebersold 1992), the idea of using a symbolic, musical scale based framework to integrate such a wide variety of possibilities into a coherent model that automatically and intimately adapts to a particular user seems a daunting exercise at best, and at worst, a scenario whose very premise hinders the simulation of truly creative behavior.

Another excellent way to assess the musical quality of what the PC model learned is to consider the degree to which its perception of the training set makes sense given the underlying harmonic structure over which the improvisations were played (and of which the learning algorithm had no explicit knowledge). In Figure 7.1, Parker's training data is presented in terms of its PC component mapping versus location in the 12-bar blues chorus. Each symbol corresponds to one of the bars in Figure 3.1 and identifies which component in Table 7.1 that BoB deemed most likely to have generated the bar's tonal content.[3]

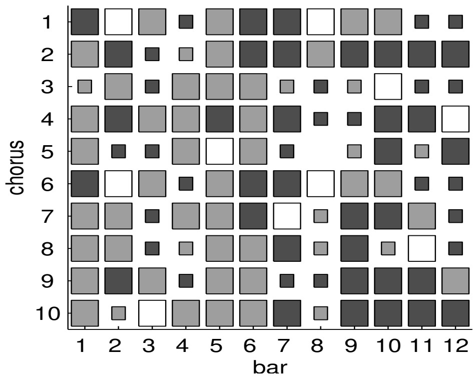


Figure 7.1: Parker's PC Modes and the 12-bar Blues

At first glance Figure 7.1 may not look very structured, but there are distinctions

[3]Two aspects of Figure 7.1 follow directly from how Parker's dataset was constructed. First, the PC sequences in chorus 1 and 6 are identical because Parker's two heads were identically transcribed (differing only by an octave). Second, the symbol for bar 8 of chorus 5 is omitted because that bar, which contained no counts, was not included in the training set.

in BoB's perception that correlate with the underlying harmonic changes that take place in the blues (Columns 1, 5, 7, 9 and 11; Section 3.2.3). For example:

- In Column 1, Parker predominantly uses the E♭ Bebop 7 scale (occurs 70% of the time).
- Columns 5 and 6 are almost universally assigned to the Bebop scales (and account for 19 out of 20 classifications).
- Only in Columns 7, 9 and 10 is the B♭ Major Bebop scale ( ) used 50% or more of the time.
- The assignments in Column 12 appropriately handle turn-backs and endings. In this key, an E♭ scale (choruses 1, 3-4, and 6-9) is a reasonable turn-back choice (Giomi and Ligabue 1988). As is typical musical practice, when Parker finishes his solos (choruses 5 and 10) he chooses a scale that returns to the tonic (B♭).

The learning priors (Section 5.7.4) ensure that any classification has the potential to be incorrect, and a solution's hardness quantifies how much error to expect (around 20% for $\Omega^{\boldsymbol{P},\mathrm{P}}$, Table 5.8). In other words, structure is not more blatant in this figure partly because 20% of the classifications are probably incorrect. Fortunately, since vMn models classification error probabilistically, it also provides estimates concerning which assignments are most likely to be correct (via the posterior, ψ); this provides a new avenue through which structure can emerge. For example, the bars in Columns 6 and 11 were much more confidently assigned to their components than were the bars in any other columns.

Section 6.4.3 presented some compelling reasons for learning the structure of more abstract musical sequences. Figure 7.1 provides a concrete example of where the training data for such a learning experiment might come from. Although preliminary experiments indicate that, at least with this level of noise, more bars of data are needed for learning to be productive, this figure motivates the types of things one might want BoB to be able to learn. Learning this temporal structure will be pursued in future work.

7.2.2 Learned Melodic Continuity and Contour

The four melodic continuity and three melodic contour components that were learned for Parker are presented in Tables 7.2 and 7.3. It is useful to consider INT and DIR behaviors together because they are linked: that unison intervals correspond to zero motions and vice versa (Section 4.3.4). Except for the fact that the bin labels for INT and DIR are different, the layout of these tables is analogous to that of Table 7.1. In particular, aside from knowing a component's mixture weight and intervallic (or directional) distribution, BoB is unaware of the 'Meaning' assigned to a component — I provided these interpretations.

Each INT component label abbreviates an interval's full name as defined in Section 4.4.4. (Both references will be used interchangeably.) Instead of showing a DIR component's twenty-seven transition probabilities, Table 7.3 presents its corresponding stationary distribution. The stationary distribution summarizes a component's asymptotic behavior (Section 6.3.3), so each bin reflects how often a particular directional history state (Figure 4.5) is likely to be occupied when parsing an infinitely long direction sequence generated by that component. For example, DIR bin label '++' indicates how likely two isolated, adjacent upwards motions are.

With respect to the meanings assigned to the intervallic components, my interpretations are motivated in part by the *Implication-Realization Model* hypothesized by Narmour (1990). Roughly speaking, Narmour hypothesizes that, at a Gestalt-based level, listening to an interval that is smaller than or equal to a Major 3rd sets up the implication that the melody will 'similarly continue,' i.e., proceed in the same melodic direction using a 'similar sized' interval.[4] He also hypothesizes that listening to an interval that is larger than or equal to a Minor-6th sets up the implication that the melody is undergoing a "reversal," i.e., pivoting around the most recent note, proceeding in the opposite direction via a 'relatively smaller' next interval. Concerning intervals within the Minor 3rd and Minor 6th range, he postulates that these "threshold intervals" are more ambiguous, having the "potential to imply either continuation or reversal, though not in equal proportion."

With respect to repeated tones (which produce unison counts), Narmour's "duplicative structure" and BoB's INT/DIR coupling are synonymous — both correspond to observing "lateral registral direction" (zero motion) and unison intervallic difference. Unfortunately, BoB's CGL representation does not currently distinguish between syncopated rhythms and repeated tones, so when ascribing meaning to this interval, I do so after having referred to its usage in specific training data parti-

[4]In this section, single quotes refer to my own choice of words; double quoted text was taken directly from (Narmour 1990).

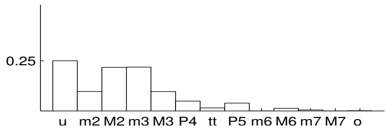
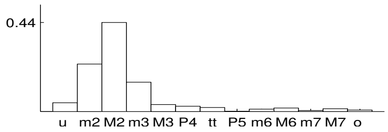
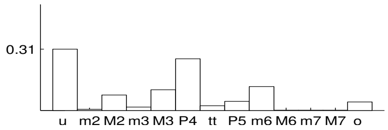
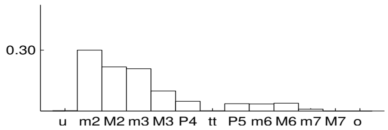
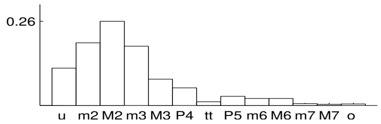
Symbol		Intervallic Probabilities	Size		π	Meaning
	1		51	42.9%	.411	relatively continuous and syncopated melodies, occassionally irregular
	2		25	21%	.229	mostly continuous, occasionally jagged, irregular, or syncopated melodies
	3		9	7.6%	.089	mostly irregular and jagged melodies, syncopation and repeated tones
	4		34	28.6%	.271	mostly continuous, occasionally jagged or irregular melodies.
			a single mixture (for comparison)			

Table 7.2: Parker's Learned INT Components

tions (Appendices E.2.6 through E.2.9). With these coarse distinctions, meanings are assigned to intervals as follows:

- *Syncopated* and *repeated* refer to specific unison interval usages.
- *Continuous* refers to intervals greater than the unison and less than or equal to a Major 3rd.
- *Irregular* refers to intervals in the "threshold" range.
- *Jagged* refers to intervals greater than or equal to a Minor 6th.

With respect to melodic contour, trends in the training data partitions (Appendices E.2.10 through E.2.12) reveal, at a high-level, three types of behavior:

1. An *upwards string* refers to a run of ascending motion.

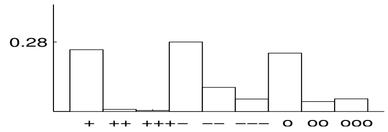
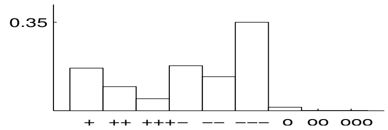
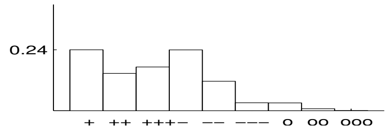
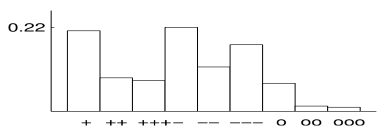
Symbol	Directional Probabilities	Size		π	Meaning
1		45	37.8%	.373	mostly syncopated/repeated tone melodies, rarely an obvious direction trend, but for an occassional downards string
2		33	27.7%	.270	mostly longer downwards strings separated by shorter upwards strings
3		41	34.5%	.356	mostly longer upwards strings separated by shorter downwards strings
		a single mixture (for comparison)			

Table 7.3: Parker's Learned DIR Components

2. A *downwards string* refers to a run of descending motion.

3. *No obvious trend* refers to a lack of presence of either of the above trends.

In the single mixture (Table 7.3), the largest proportion of probability mass lies on states that record downwards motion ('–', '––', '–––'), which makes sense given that:

> "anyone interested in melodic analysis notices right away the phenomenon of ascending leaps followed by descending linear patterns." (page 220)

With respect to Parker, this trend has also been commented on in Hörnel, Langnickel, Sandberger, and Sieling (1999). Roughly speaking, in the learned $\mathsf{k} = 3$ model, BoB will assign bars that exhibit this behavior to the component. Although overall, when contrasting the different components relative preference for ascending versus descending trends, it is clear that downwards motion is still most prevalent, the component accounts for the smallest portion of the training data.

Instead, the most popular component, , corresponds to melodies that display no obvious directional trend. This lack of trend is largely due to the bars in this partition containing more rests and larger intervals than are found in the other two partitions. As a result, the segments in ■ tend to be more jumpy and/or contain melodic lines that divide into more than one phrase. It is quite possible that, with a better segmentation scheme, a larger portion of the training data would then be explained by a downward string-based component(s). Regardless, it is not the case that what is globally the most prevalent in a training set should necessarily produce the most heavily weighted component. While this may seem counter-intuitive, it is entirely appropriate for a melodic model. For example, only such a model could handle the case where overall more motion is downwards, but the motion occurs in a highly concentrated, small number of segments.

Where Narmour (1990) applies his theories to specific melodies, probabilistic vMn components must be interpreted in likelihood-based terms — in terms of what behavior is most and/or least expected. Roughly, I quantify this notion with the following adjectives, listed in order from most to least likely: *mostly*, *relatively*, *somewhat*, *occasionally*, and *rarely*. In general, such quantifications need to be considered as a matter of *degree*, i.e., in relative as opposed to absolute terms. For example, reasons such as voice-leading mean that melodic intervals will tend to be relatively small in the first place, which is why, in the single INT mixture (Table 7.2), a majority of the density lies between the unison and the Perfect 4th. As such, when I say that the INT □ component is 'mostly irregular and jagged,' this 'mostly' is a matter of degree.

Finally, let me clarify why Narmour's model received so much attention in this section. BoB's primary purpose is *not* to encode Narmour's ideas, although I do believe that many of them could be used to improve BoB's melodic feature space. To seriously implement his ideas, BoB's representation would need be to be extended so as to: consider rests; treat the octave interval separately from larger-than-octave intervals; consider differences between pairs of intervals; etc. In addition, while Narmour distinctly separates intervallic and directional trends, his theory does consider various combinations of these behaviors, and so some form of additional coupling between intervals and melodic direction would be needed. Interestingly, the representation presented by Steinsaltz and Wessel (in progress) explicitly handles two of these issues (pairs of intervals and their melodic directions). In concentrating on those aspects, however, their model, which lacks the ability to observe any trend longer than a pair of intervals, does not provide as rich a view of melodic contour as does BoB.

7.2.3 Conglomerative Perception Examples

In this section, several sets of examples from Parker's training data, each set mapping into a different playing mode, are presented, contrasted, and compared to provide a more concrete idea of how the vMn perception mechanism works when perceiving short segments of melody.

These examples are listed in Table 7.4, each row presenting a different playing mode, as indicated by the bars' *conglomerative class tags*.[5] For example, the playing mode associated with the bottom row is $y^{\mathrm{C}} = \langle 1, 4, 3 \rangle$, and this playing mode's components can be interpreted using Tables 7.1 through 7.3 (e.g., the first row in Table 7.1 is the PC mode's generator, the fourth row in Table 7.2 is the INT mode's generator, and so on). Below each example, its bar location within the training set (Appendix E.1) is given. The reason that a pickup note into each bar is displayed is because the INT and DIR histograms rely on this information (via MRP). When classifying an example's PC histogram, pickup notes are ignored.

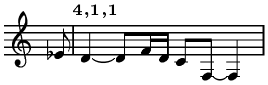
Example 1	Example 2	Example 3
 bar 31	 bar 91	 bar 102
 bar 28	 bar 36	 bar 65
 bar 48	 bar 59	 bar 94

Table 7.4: Parker Conglomerative Class Examples

When people hear these examples in demos, they tend to appreciate

[5]This tag is comprised of the three bold-face, comma-separated numbers printed above and to the left of each bar. From left to right, the numbers in this tag indicate which PC, INT, and DIR components BoB deemed most likely to have generated the example.

the fact that, within rows, the examples sound more similar than they do between rows. This observation makes sense given that listeners can perceive various aspects of a melody's tonal, intervallic, and contour properties (Deutsch 1999; Krumhansl 1990; Bartlett 1993). However, when listening to these segments, it is difficult to pinpoint exactly *why* these distinctions are perceived. It is rather a transient, vague impression. Because the vMn model is a precise specification, its underlying mechanisms can be used to help shed light on these distinctions. To really appreciate these examples, on must *listen* to them. Even though on the surface — for example, with respect to an edit-based similarity view — all of the segments look quite different from one another, some of them just "fit together" more with one another than with others.

7.2.4 Quantifying Per Row Trends

Per-row, examples can be compared in terms of how many features are *shared* or *absent*, i.e., which features across examples contain or do not contain counts? Similarly, *partially shared* features are those whose counts are obtained from two of the three examples. Per-row, features can also be compared in terms of their *frequency of occurrence* (FoO), a term that is best defined by way of an example. Consider row two, which has a total of nineteen PC counts. If one were to ask how many of these examples' counts lie on bins C and E♭, the FoO would be 2:1:16 because there are two counts on bin C (bar 28 and bar 36) and one count on bin E♭ (bar 65); the sixteen remaining counts are unclaimed.

These metrics can be used to loosely quantify the commonalities between examples,[6] from which a reader can proceed to reason about the most notable differences between rows. Since the components in Tables 7.1 through 7.3 were used to partition these examples in the first place, it is no surprise[7] that the commonalities found are consistent with the results reported in these tables.

7.2.5 Parker CGL Examples: $y^{\mathrm{C}} = \langle 2, 2, 2 \rangle$

In the examples in row one, except for the A in bar 91, **pitches coincide with the E♭ Bebop-7 scale.** Pitch classes C, F, G, A♭, and B♭ are shared, while E, G♭, and B

[6]Bold type is used to highlight the coarsest of similarities.

[7]Rather, this had better be the case!

are absent. **Melodies are also essentially continuous**, with shared intervals m2 and M2, and partially shared intervals m3 and M3 (the only other present intervals are a single P4 in bar 31, a M7 in bar 91, and a m6 in bar 102). Furthermore, **melodies are mostly descending runs.** More specifically, these examples produce the following direction sequences (equation 4.1):

$$\begin{aligned} d_{n=31} &= \langle -, +, -, -, -, -, +, +, -, - \rangle; \\ d_{91} &= \langle +, +, -, -, -, -, +, +, +, -, -, -, - \rangle; \\ d_{102} &= \langle +, -, -, -, -, -, - \rangle. \end{aligned} \tag{7.1}$$

As a result of how these sequences are parsed (Figure 4.5), the following transitions are shared:

$$-/-,\ --/-, ---/-\,.$$

Present transitions, which delineate the other melodic motions used to connect these descending runs together, are:

$$+/-, ++/-, +++/-, -/+, ---/+, +/+, ++/+\,.$$

The other remaining 16 transitions are absent.

7.2.6 Parker CGL Examples: $y^{\mathrm{C}} = \langle 4, 1, 1 \rangle$

For the examples in row two, **pitches coincide with the E♭ Major scale.** Although only pitch-class F in this scale is shared (D is noticeably absent in bar 65), all but the reasonably probable B♭ pitch class are present. **Almost all of the intervals are continuous or syncopated**: the FoO for syncopated, continuous, and irregular intervals is 5:11:3. Although u is the only shared interval, M2 through M3 are partially shared. These segments also **do not display an obvious trend in melodic direction.** Aside from the short descending runs in bar 28 and bar 36, only single steps in particular directions occur. The only shared transition among these three is $+/-$.

7.2.7 Parker CGL Examples: $y^{\mathrm{C}} = \langle 1, 4, 3 \rangle$

For the examples in row three, **pitches coincide with the B♭ Major Bebop scale.** All of the tones in this scale are shared except for pitch-class F and A. The G♭ tone

— which is what distinguishes this scale from the more traditional B♭ Major scale — is noticeably present.[8] **While melodies are generally continuous, occasionally a jagged interval appears.** Shared intervals include m2, m3, and M3. Viewed as one higher-level meta-feature, the m6/M6 combination is also a shared feature. This combination of intervals is an important part of these examples' character, and also one reason why each maps into component four (as opposed to component two). Except for bar 94, **these melodies contain more ascending than descending runs.**

7.2.8 Example Bar 65 Revisited

It is revealing to take a closer look at how the INT histogram for bar 65 is classified because, in this example, the mixture model's weights play an important part in its classification. This example is further detailed in in Table 7.5. The bar's intervallic feature vector (x^{I}) is displayed, as is its assigned class (the *winning* component) and that component that was the next most-likely generator (the *runner up*). These components' posteriors (ψ^{I}) and surprises (labeled likelihood l^{I}) are also shown.

With respect to the posterior, component one is the clear winner (0.63 versus 0.37). However, if one were to only look at the histogram's surprise under both models, component two "better explains" the histogram's counts (0.0017 versus 0.0016). This example is a good one to look at in terms of understanding the vMn model's classification scheme because, although the u and M3 terms on component one are larger, these each only account for a single count in x^{I}. Component two's M2 interval, on the other hand, is slightly more probable than component one's, but this bin explains four counts. What the surprise values indicate is that if one were to *only* look at the probability of the Multinomial, then component two's average is closer to x^{I} than is component one's. Because component two is approximately half as likely as component one ($\pi = 0.23$ versus 0.41), bar 65 is otherwise assigned.

7.3 Customized Perception: Stephane Grappelli

In this section, the appendix that details the user model learned for Stephane Grappelli's training data, $X^{\boldsymbol{G},\mathrm{C}}$, is described. Some conglomerative examples are also

[8] G♭ is also noticeably absent in row one.

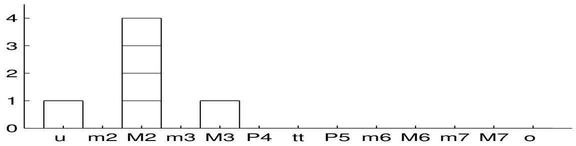
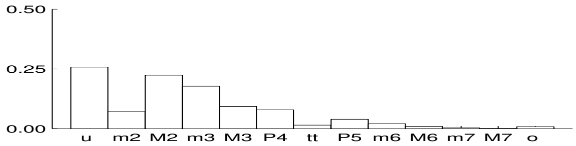
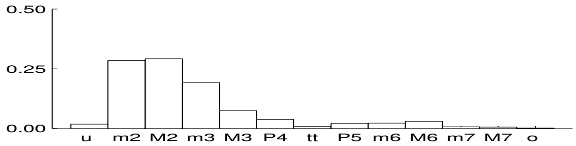
bar 65	
INT histogram x^{I}	
winning component $y^{\mathrm{I}} = 1$ $\pi = 0.41$	$\psi^{\mathrm{I}} = 0.63 \quad l^{\mathrm{I}} = 0.0016$ 
runner up component $y^{\mathrm{I}} = 2$ $\pi = 0.23$	$\psi^{\mathrm{I}} = 0.37 \quad l^{\mathrm{I}} = 0.0017$ 

Table 7.5: Parker INT Classification Example

presented. For the score used to construct this training set, see Appendix F.1. This model is the same one that was first presented from a machine learning point of view in Tables 5.8 and 5.10.

Analogous to what was just presented for Parker, Appendix F presents information concerning Grappelli's learned components and their perception of his training data. For a high-level summary concerning what musical information Grappelli's PC, INT, and DIR components encode, consult Tables F.1 through F.3. This appendix also lists how Grappelli's model segregates his improvisations and displays graphs that demonstrate how these models perceive playing mode as a function of underlying harmonic context.

Several sets of examples from Grappelli's training data, each set mapping into a different playing mode, are demonstrated (Table 7.6) to provide some concrete examples of how his modes differ. This table's setup is analogous to the one presented for Parker (Table 7.4). The only difference is that conglomerative class components are defined in Tables F.1 through F.3 and bar location is with respect to the training set defined in Appendix F.1.

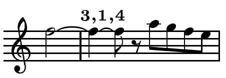
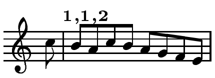
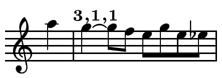
Example 1	Example 2	Example 3
 bar 3	 bar 88	 bar 117
 bar 23	 bar 28	 bar 119
 bar 54	 bar 92	 bar 120

Table 7.6: Grappelli Conglomerative Class Examples

7.4 Grappelli CGL Examples: $y^{\mathrm{C}} = \langle 3, 1, 4 \rangle$

In the first row, **tones correspond to a fragment of pitch-classes in the F Major scale**, ranging from the leading tone (E) up through the scale's 3rd degree (A). All of these pitch classes are shared but the A in bar 3. **These melodies are continuous and heavily syncopated**, shared intervals include: u, m2, and M2. m3 and M3, also continuous intervals, are present, as is the fairly surprising two-octave leap (bar 88). When **syncopation** is not controlling melodic contour, **descending runs are the main directional trend.**

7.5 Grappelli CGL Examples: $y^{\mathrm{C}} = \langle 1, 1, 2 \rangle$

In the second row, with the exception of the B, **all shared and present pitches coincide with the F Major scale.** It is interesting that B is shared because it is also the scale's tritone, an especially important tone in jazz. Absent tones include D♭, E♭, G♭, and A♭. **These melodies are primarily continuous,** although not syncopated. The only large interval is the single M7 leap in bar 28. The only syncopated interval is in bar 119. Given that the FoO for unison, m2/M2, and all other intervals is 1:21:1, it may seem surprising at first glance that INT component one was chosen, for it also has a reasonable sized unison probability. It is these examples' heavy use of the m2/M2 intervals that attracted them to this component. More generally, component one's large m2/M2 probabilities — very common melodic intervals — are a primary reason why its mixture weight is so large. In terms of direction, **descending runs are overwhelmingly present.**

7.6 Grappelli CGL Examples: $y^{\mathrm{C}} = \langle 3, 1, 1 \rangle$

The mode of the third row's examples was chosen to demonstrate how different segments can look when just one of a CGL's subgoals is changed. In particular, with respect to PC and INT, these examples are almost identical in nature to row one. The only difference is that row three's examples display no obvious upwards or downwards trends. Instead, directional content is a mix of syncopation, up/down steps, and short ascending or descending runs.

7.7 Listening Experiments

Two different types of listening experiments were conducted, each designed to answer a specific question. In the *playing mode variation* experiments (PMV), human subjects were asked to listen to pairs of segments and assess whether or not they thought the two were generated by the same or different playing modes. The hypothesis in this experiment was that listeners would be able to tell the difference between same and different playing modes, thus justifying the need for a mixture model that can provide multiple (versus a single) components. In the *learning model variation* experiments (LMV), subjects were asked to first listen to: i) a call from a particular musician's training set, and ii) a pair of responses to this call. Listeners were then asked to assess whether or not each of these responses was generated using the same or different learned models. Once again, the hypothesis in this experiment was that listeners would be able to tell the difference, justifying the need for user-specific learning.

7.7.1 Generating Pairs of Solos

Individual pairs of generated responses (solo one, s_{r}, and solo two, $s_{\mathsf{r}'}$) were obtained using a generate function very similar to `vMnGenerate` (Table 6.6). Whereas `vMnGenerate` performs solos in real-time, the generator used in these experiments calculated solo pairs (s_{r} and $s_{\mathsf{r}'}$) in advance using a particular pair of call contexts (r and r'respectively). A single *listening trial* is made up of one pair of solos. A trial was played back to the user through a simple keyboard-driven menu, allowing listeners to repeat the playback of each solo pair, querying them to make an assessment concerning the pair's sameness or difference, and logging their assessment.

7.7.2 Experimental Setup

One-on-one, I would introduced a listener to BoB, after which they would first perform a PMV experiment and then perform one (or two) LMV experiments (depending upon how tired they were after the first LMV experiment). Eleven adults listened, most of whom were affiliated with the Computer Science Department at Carnegie Mellon University. About three-quarters of the listeners were proficient improvisors. It usually took a listener about an hour to finish the experiments.

Playing Mode Variation Specifics

With PMV, Parker's learned model and training data were used to construct listening pairs. Each solo occupied one bar, containing on average about 9 unique pitches. Solos were played back at a tempo of 150 beats-per-minute, requiring about 1.5 [sec] of time per solo. Since each pair was played back with a short pause separating each solo, about 3.5 [sec] in total was required to listen to a complete trial. Twenty-five pairs were generated, thirteen sharing the same mode and twelve having different modes. The random-seed chosen by the first listening subject was used to initialize the generator for all subsequent listeners, i.e., all listeners heard the same response pairs. For a transcription of BoB's responses, and more detail concerning generation, see Appendix G.1. Eleven different listeners completed this experiment.

Learn Mode Variation Specifics

LMV experiments were either *Grappelli-based* (or *Parker-based*), depending upon which musician's training data was used to simulate the call contexts. Once a particular musician was chosen to be the basis, the first response was always generated using the learned model for that musician, e.g., generation was driven by the Grappelli (Parker) model. What varied between trials was whether or not the same (versus the other) learning model was used to generate the second response. While the call impacts a response in that it is mapped into an abstract goal (mode and surprise), the learned model's impact on a response is twofold (it defines this abstract mapping and provides the component parameters used to drive generation).

Call contexts were between three and four bars long, each response containing on average about 22 unique pitches. A tempo of 175 (150) beats-per-minute for Grappelli (Parker) was used for playback. Listening to a Grappelli (Parker) response took on average about 5 (6) [sec]. Each call/response pair was separated by a short pause. In total about 22 to 26 [sec] of listening per trial was required. Fifteen pairs were generated, eight sharing the same model and seven using different models. Once again, all listeners heard the same pairs. For a transcription of BoB's responses, and more detail concerning generation, see Appendices G.2.2 and G.2.1.

Some listeners were too tired to do two LMV experiments. In these cases only the Grappelli based experiment was run. Ten (seven) listeners completed the Grappelli (Parker) based experiment.

In the LMV experiments, the listener's goal is subtly different than it is in PMV. Essentially, the listener is being asked to distinguish between whether or not the same call is passed through a same or different "generative filter," where the filter's operation is influenced by two somewhat different operations. When two different models are used to generate responses to the same call, one might naively expect the differences between these models to blatantly emerge in the resulting responses. However, realize that because the *same call is input to both generators*, each will map this into that goal mode that most closely encodes its pitch-sequence behavior given that that model. To some extent, this mapping attenuates some of the differences that actually exist between the models.

Introducing Listeners to the System

Listeners conducted the experiments themselves, listening to audio generated by a PC[9] in my graduate student office, entering their answers and progressing to the next trial via a keyboard-driven menu. To order to avoid influencing the listener's decisions in other ways, solos were played back mechanically using a simple MIDI synthesizer patch.[10]

Before listening began, I would briefly summarize the purpose of the experiment: to distinguish between "sameness" and "differentness." I would also describe at a high-level how BoB operated and how a typical improvisor would likely interact with the complete system. Since the goal of these experiments was to assess BoB's handling of pitch-sequence content, listeners were informed to "try and ignore rhythm as much as possible," rather focusing on the solo's pitch-based content (tonality, general shape, etc.). Several participants told me that the most helpful piece of information they received during this introduction was to keep in mind the following question:

> If you were interacting with BoB and it had just played solo $s_{r'}$ instead of solo s_r, would this have had a "similar" impact on your improvisational experience?

Before a set of listening trials began, the user was given a couple of demonstration rounds, so that they could practice listening to response pairs, drawing conclusions about them, and entering their assessment into the computer. During this demo, listeners did not know whether a pair was different or not in advance, but after entering their assessment BoB would tell them what the answer was. The point of this exchange was to allow the users to gain some practice paying as much attention to pitch-based content as possible, providing some concrete interaction to help them acquire a notion of what was being asked of them. While this procedure ran the risk of biasing a listener (in the worst case, "teaching them" to distinguish between various components in the mixture model), no listener felt compelled to run more than one or two of these demonstration pairs. Given that a listener's success rate usually improved as the trials progressed (because, in an unsupervised fashion, they were able to develop a better idea of what it meant to distinguish between two concepts), I believe the biasing impact of the initial demo trail was slight.

[9] Audio was produced using a *SoundBlaster* sound card.

[10] One disadvantage of this playback was that the music had a primitive "*Donkey Kong*" sounding video-game quality.

7.7.3 On the Subject of Listening to Music

I cannot emphasize how difficult the task I was asking listeners to perform is, and to all who gave their time, I am grateful. Ideally, a listener's assessment should be based upon their instantaneous, gut response to a pair of solos. However, asking someone to assess sameness or difference makes it more difficult to obtain an instantaneous reaction. Rather, aware that they are being asked to perform a specific task, listeners will invariably attempt to organize what they hear as the experiment progresses in order to make the task as easy as possible.

Another issue concerning data collection via music listening arises in virtually all music-cognition experiments. Often, studies are run with listeners hearing a very short series of tones that occupy a much smaller time span than was used in each LMV trial. While very short listening segments are useful because they allow one or two variables to be carefully controlled and critically listened to, the problem is that these conditions do not reflect the richness of what goes on when people listen to an entire musical experience in real-time. Unfortunately, music listening is difficult to control and monitor in more complex settings because melody is so heavily influenced by context. Another problem with melodic listening is that one or two "bad notes" can stick out more than the other twenty-one or so good ones. Music listening is a temporal phenomenon — unlike a painting, its content can not be perceived all at once. Without being able to instantaneously appreciate the whole experience, it becomes questionable to what extent, and under what conditions, listeners are assessing what one really wants to measure.

Suffice it to say that music listening tests are difficult to conduct and interpret. This fact is especially true in IMC domains, where the goal is not so much to assess *how* people listen to music as to assess to what degree the system *enhances* their improvisational experience.

7.7.4 Experimental Results

The results for the Parker PMV test and for the Grappelli- and Parker-based LMV tests are summarized in Table 7.7. Results are segregated according to the listener's ability to identify same and different pairs. Histograms for each experiment and segregation provide a more detailed look at the data that was collected (Table 7.8).

Each histogram count corresponds to a listener's average *performance*, i.e., his

Experiment			Same Correct					Different Correct				
		n_{subj}	Ave	95% CI		n_{trials}	P-Val	Ave	95% CI		n_{trials}	P-Val
PMV	Parker	11	**0.69**	**0.61**	**0.78**	13	**0.001**	**0.64**	**0.57**	**0.71**	12	**0.003**
LMV	Grappelli	10	0.61	0.50	0.72	8	0.055	**0.91**	**0.84**	**0.98**	7	**0.000**
LMV	Parker	15	0.51	0.28	0.74	8	0.47	0.44	0.23	0.64	7	0.72

Table 7.7: Listener Results

or her ability to correctly assess n_{trials} of sameness (or difference) pairs. The results in Table 7.7 quantify expected listener performance. Our concern is to determine whether or not listeners can distinguish between same or different pairs with better than random-guessing performance. In this table, performance is normalized over the total number of subjects (n_{subjs}). The Student's t test is used to calculate P-Values with respect to the following research hypothesis (Casella and Berger 1990):

> Listeners can identify sameness (or difference) between pairs with better than 50% accuracy.

Those experiments in which this distinction is at least 0.05 significant are highlighted in bold type. The 'CI' field refers to the 95% confidence interval of the estimated listener performance rate.

Especially for the LMV tests, it would have been better to increase the value of n_{trials} in order to ensure that the underlying distribution of the observed listener averages is approximately Gaussian.[11] However, due to the time required to take the tests, collecting more data would have been impractical. **Nonetheless, there is little doubt that in the LMV Grappelli-based setting, listeners can tell when the Parker model is being used instead of the Grappelli model.** In this case, the 95% confidence interval for how well users can *distinguish between different models* is exceptionally high ($[0.84, 0.98]$). When Parker's solo calls were being responded to with Grappelli's model, distinctions between models were less obvious. This is a result of the fact that although Parker's playing preferences (and learned model) are more "far out,"[12] when such playing is mapped onto Grappelli's model, the call's "wildness" is somewhat attenuated. In general, listeners seemed to have more trouble identifying similar LMV trials than they did identifying dissimilar

[11] The Student's t test, which is based on exact sampling theory (Spiegel 1994), alleviates a similar concern about the size of n_{subjs}.

[12] For example, realizing more dissonances and syncopation.

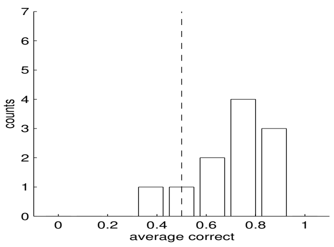
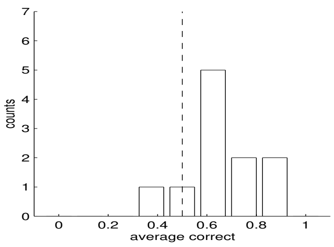
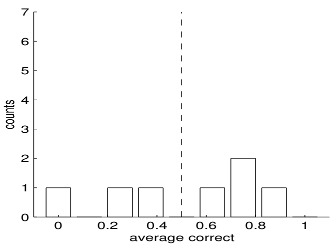
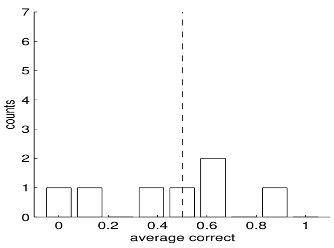
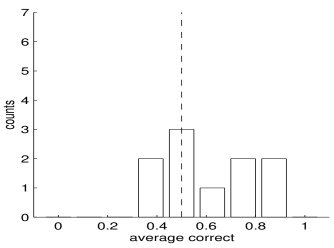
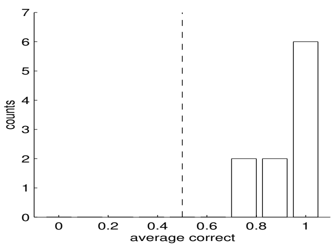
Same Assessment	Different Assessment
PMV Experiment; Parker Based	
	
LMV Experiment; Grappelli Based	
	
LMV Experiment; Parker Based	
	

Table 7.8: Listening Experiment Results

ones. Within a given model "similarity" is still coarsely defined (in part because of sample size sparsity), which offers one explanation for why dissimilarity is more easily distinguished in general. For an IMC, this coarseness is beneficial, providing rich generalizations in situations where the computer must at least *appear* creative.

Listeners were also able to significantly distinguish between same and different Parker playing modes, sameness being slightly easier to identify than difference, e.g., 69% of same (versus 64% of different) correctly identified on average.

With the PMV pair solos shown in (Appendix G.1, the interested reader can further investigate how generation works. For example, it is instructive to compare the bars generated between two different and same pairs, constructing histograms for their CGL features and comparing the shapes of these histograms to the shapes of their underlying generative component parameters. Through such an analysis, the richness of the vMn generalization mechanism and its inherent reliance upon sample size sparsity is further demonstrated.

Some listeners observed that most of their decisions were influenced by intervallic and directional trends. In particular, rhythmic syncopation and very jagged melodies were often noted as decisive factors. Some listeners complained that there was no harmonic background provided for these listening experiments, hypothesizing that had harmonic knowledge been provided, distinctions between different tonal modes would have been easier to make. In general, each subjects' running commentary during their listening trials was informative (I have recorded these comments in a log book). While these logs reaffirm my suspicion that such vague listening tests are difficult to administer and interpret, they also suggest mechanisms for future improvement. For example, about half the listeners noted LMV pairs were simpler to assess than PMV pairs. This observation makes sense given that relative distinctions (e.g., compare two responses to a particular call) are typically easier to make than absolute distinctions (e.g., determine if two modes are identical or different). However, because of the length of an LMV trial, only one or two listeners were able to carefully remember and recall what they had heard within it. As a result, other listeners found the PMV trials easier to assess.

Unfortunately, there is likely no procedure that will please all listeners. For example, while some complained that last note clamping made assessment difficult because it diminished the difference between two solos in their minds, others complained that in one pair the last note "did not fit as well," making assessment easier. While some people were extremely frustrated by the lack of harmonic background, other people seemed to enjoy listening to melodies in this less constrained context. **What was most encouraging about these experiments were that in all but two cases, musicians found the system's interactive behavior both interesting and rich, regardless of the bad note or two it would often play.** To summarize the general impression musically proficient listeners seemed to have about BoB's performance:

> Wow....it seems like the system is *really trying* to respond and interact with the call. Its definitely a far cry from random behavior...

7.7.5 With Respect to Solo Trading

I will close this chapter by calling attention to the solo trading scores in Appendix H. These scores outline the live demos that I typically play when giving talks (MIDI files of these scores will be available on my web page, `http://www.cs.cmu.edu/~bthom`, shortly). Four different solo-trading scenarios are presented:

1. Parker trading solos with BoB trained on Parker;
2. Parker trading solos with BoB trained on Grappelli;
3. Grappelli trading solos with BoB trained on Grappelli;
4. Grappelli trading solos with BoB trained on Parker.

In these scenarios, BoB and the musician trade fours. Four bar calls were taken from contiguous segments of the musician's training data (every last bar was replaced by a rest so that there was a short pause between each call and response). To generate a response, BoB would transform this call context as described in Section 6.4.2, producing three bars of response (and a rest bar) for the next four bars of accompaniment.

Because these demos generate more bars than an LMV trial, audiences typically have an easier time appreciating the distinction between these different scenarios than they do distinguishing between pairs in a particular LMV trial. What is interesting about this observation is that short-term, gut reactions become less important when listeners collect more data, forming longer-term opinions about what the agent is doing. In other words, in longer-term interactive contexts, the importance of an agent's ability to customize itself to a user and respond to various playing modes may be even more important than the types of listening experiments conducted above indicate.

Chapter 8

Related & Future Work / Limitations

The IMC domain is interdisciplinary and draws on techniques from a wide variety of fields: real-time systems, artificial intelligence, human-computer-interaction, statistics, music analysis, psychology and cognition, etc. In this chapter, a few related research projects in interactive computer music systems, melodic representation, and machine learning are presented. Since related research naturally sets the stage for considering limitations of this thesis' approach and motivates directions for future research, these topics are also presented.

8.1 Real-time Interaction

As a necessary precursor to live interaction, BoB needs rhythmic quantization (RQ). **The whole point is to have musicians play *with* BoB. This issue is the most pressing future research item.** The difficulties associated with RQ were presented in Sections 3.1.2 and 3.1.3.

Given its strong probabilistic basis, the RQ technique proposed by Cemgil, Desain, and Kappen (2000) is a natural choice for integration with BoB. Their approach is Bayesian, combining a probabilistic code-book that models the likelihood that particular codes (scores) "explain" a performed onset sequence with a prior that penalizes "complex" codes more heavily than "simple" ones. This penalty makes the classification task well-defined because it ensures that more complex scores will not necessarily better explain performances. BoB's requirement for encodable rhythms can be handled by restricting the contents of the codebook.

An attractive aspect of their model is that its explicit mathematical formulation made it straightforward to tune parameters using perceptual data. (This data measured musicians transcribing and perform various rhythm scores (Desain, Aarts, Cemgil, Kappen, and van Thienen 1999).) An open question is how well their model will work in live improvisations, where higher-level musical context may have a large impact on a musician's expressivity. Since BoB can adapt itself to whatever input it receives, in practice this issue may be irrelevant.

8.2 Musical Expression

An obvious limitation is BoB's mechanical, unexpressive playback. In addition to significantly bolstering a system's believability, expression plays an important part in communicating higher-level intent (for example, emotional meaning (Bresin and Friberg 2000)). With the recent strides that have been made in automating musical expression, integrating BoB with one of these technologies is worth considering.

In Dannenberg and Derenyi (1998), synthesis and low-level performance models were integrated. Their research focused on standard trumpet-based performance practices and required less symbolic knowledge than other methods. While integrating this model into BoB would require some modification (their performance model was tuned for classical music, required phrasing information, etc.) the endeavor looks promising.

Another possibility is the *Saxex* model (Arcos, de Mantaras, and Serra 1998), which uses case-based reasoning techniques to find synthesis parameters that are "appropriate" for expressing a score. Integration with BoB is straightforward in that *Saxex* operates upon note sequences in specific harmonic contexts. More difficult issues remain. For example, *Saxex* does not run in real-time and the current system operates upon "expressivizing" an audio signal (versus MIDI). Fortunately, providing an interface that would allow the system to operate directly with MIDI data is part of Arcos, de Mantaras, and Serra (1998)'s research agenda. Given that *Saxex* was crafted to operated on jazz ballads, the interesting research question is how well their approach will perform in more general improvised settings.

8.3 Melodic Representation

There are obvious limitations to BoB's melodic representation. For example, no effort has been made to *explicitly* craft what notes within a segment are most salient. This task, however, boils down to developing "a *really* good" melodic representation. As with any complex human domain, this development requires one to essentially fulfill the "AI dream." Nevertheless, the search for better melodic representations lies at the core of computer music research for:

> How do we make sense of what we hear *as music*? One of the most evident features of [a] piece is that it has a melody — in some respects it *is* a melody. (Cross 1999)

In terms of computational modeling, two often cited sources of inspiration are Narmour (1990) and Lerdahl and Jackendoff (1983), which provide details about how one might model the effects of rhythm, meter, pitch, melodic contour, etc. As insightful as these theories are, they are only partially useful for creating a computational melody model. Ultimately, there are just too many different factors that influence melodic understanding. For example, with respect to the "'brute' mechanistic system" proposed in Narmour (1990), the author cautions:

> Let me, however, emphasize as strongly as possible that ... one cannot mechanistically apply [...it...] in the explanation and interpretation of melody. For idiostructural context or style itself always impinges upon [...it]. That is why I say that [...it...] constitutes a hypothesis: one must not invoke it simplistically.

With these difficulties in mind, extensions which address some of BoB's representational shortcomings are proposed.

It is worth investigating how much a "more musical" segmentation scheme would improve performance. The work of Lerdahl and Jackendoff (1983) introduced a foundation upon which many real-time segmentation algorithms have been constructed (Rowe 1993; Stammen and Pennycook 1994). Recently, Cambouropoulos (2001) proposed LBDM, a computational model for segmenting melody that is simpler and more general than other models based on a limited set of rules, e.g., (Lerdahl and Jackendoff 1983). A crucial aspect of LBDM is that it can be applied to non-quantized performance data.[1] Since BoB handles the number of events per segment as variable, its integration with LBDM is straightforward. LBDM's functionality would allow interesting quantitative (e.g., are clusters better separated?) and qualitative (e.g., musically, do the clustering partitions make more sense?) questions to be explored.

I also expect that a solo's quality may dramatically improve if knowledge concerning key notes (like the target notes considered in Toiviainen (1995)) is incorporated into the generation scheme. Another possibility is to investigate reducing the number of "really bad notes" that are generated by experimenting with various harmonic and metrical constraints. When live interaction is in place, a fruitful way to begin improving the system's melodic representation will be to have as many musicians play with the system as possible. By carefully analyzing the corpus of interactions that result, a comprehensive list of "serious errors" can be compiled. This list can then be used to incrementally address BoB's melodic shortcomings by: adding new features or modifying existing features, adding new constraints, either manually or with the help of additional ML algorithms, etc. Such incremental improvements will also shed light on other important issues. For example, as constraints are added, more information about how these constraints impact `searchPS`'s performance will be obtained.

8.4 Interactive Computer Music Systems

Rowe (1993) observes that "interactive music systems change their behavior in response to musical input" — providing a level of companionship to the musician by definition. Prior work tends to fall into one of two camps (Walker 1997). The first camp embraces the author-the-aesthetics paradigm, where the composer is the author of the program, and it is assumed that he or she will set it up to respond "appropriately." In the second camp, the attempt is to build an aesthetically neutral system that can be tailored by an end-user. (The *Max* program is a good example of this approach (Puckette and Zicarelli 1991; Winkler 1998).) With the advent of machine learning, interactive systems have also begun to employ learning techniques for operationalizing more autonomous musical aesthetics.

[1] Even when a system ultimately operates on a transcription, it is reasonable to expect that segmenting must operate on raw data.

8.4.1 Roger Dannenberg

Dannenberg's artistic goal is to have the computer be an extension of his compositions. Many of his interactive implementations fall into the author-the-aesthetics camp. Typically, his compositions make real-time performance decisions that adapt to the nuances of a human performer. Music recognition is either hand-coded and/or trained using human-supervised learning data. Music generation is generally designed to carry out the composer's (rather than the performer's) goals.

Perhaps BoB's PC modeling scheme and the scoreless accompanist developed by Dannenberg and Mont-Reynaud (1987) are most related. The purpose of this system was to accompany a soloist while they freely improvised over a 12-bar blues (tempo was not restricted). Before live interaction began, MIDI training data was collected while the improvisor soloed over a computer-provided blues progression. This data was used to build a real-time model that could predict at what location in the blues progression the musician was likely to be soloing over. A highly local view of the blues was constructed — a different pitch class distribution built for each eighth-note — and for each eighth-note, a single Multinomial distribution was estimated. It was noted that a less disperse likelihood function would probably improve performance and the vMn model provides one mechanism for addressing this issue.

Another motivating piece of work was the improvisational style recognizer developed by Dannenberg, Thom, and Watson (1997). In this research, improvisations were broken up into segments and converted into histogram-based features. Improvisations were collected in a supervised fashion: the computer told the musician what type of improvisation to play next (e.g., bluesy, frantic, etc.) and the musician, motivated by this suggestion, improvised freely. This data was used to build a classifier that could distinguish between different improvisational intentions. Since classification performance was impressive, the natural next step was to consider using this technology in a more directed fashion, e.g., to have the computer respond in kind. This consideration poignantly demonstrated the difficulty in closing such an abstract learning loop and motivates my interest in melodic generation.

8.4.2 Robert Rowe

Like Dannenberg, Rowe has extensively experimented with creating and performing with interactive computer music systems. His *Cypher* system was developed on top of a strong musical foundation, providing a variety of meaningful (hand-tuned) feature detectors that provide classifications about: loud versus soft, high versus low tones, phrase detection, chord identification, etc. *Cypher* can operate in both scored and improvisational settings. The underlying architecture was based on Minsky's society of mind ideas (Minsky 1986; Minsky 1981) — individual tasks were handled by rather simple agents that worked in isolation. In terms of using Cypher, the paradigm is author-based:

> The user's task is to configure the way in which the player will respond ... according to connections made by the user between features and transformations.

Rowe's sensibilities fit naturally into the IMC domain. A large part of my vision was spelled out in Rowe (1993):

> In developing my own computer musician, I attempt to make human participation a vital and natural element of the performance situation...[to]...develop computer musicians that do not just play back music *for* people, but become increasingly adept at making new and engaging music *with people*, at all levels of technical proficiency.

Implicit in his approach is the pragmatic observation that the audience is be willing to suspend their disbelief (page 43). In particular, although Rowe finds the task of building an interactive computer music system daunting because the content of the representations and the goals are roughly equivalent to human musicianship, this does not deter him from arguing that:

> even a little bit of musicianship can make computer programs engaging partners for a variety of applications and compelling members of performing and improvising ensembles. In other words, not all musicianship has to be encoded to make these systems useful, and many incremental improvements are available to take them beyond the level they have already reached. (Rowe 2000)

8.4.3 David Wessel

Open-ended improvisational experiences are actively sought in Wessel's computer-assisted performance settings, which, at their core, embrace an author-the-aesthetics paradigm, albeit in a more spontaneous fashion. For example, the emphasis in Wessel, Wright, and Kahn (1998) was to create technologies for organizing and controlling the access to rhythmic and timbral material so that human computer musicians could author meaningful musical experiences to accompany the performance of a classical *Khyal* singer on-the-fly. Typically, Wessel's interaction with live improvisers involves performing alongside them in a meta fashion, directing their live acoustic or MIDI data into internal software modules that have been constructed with the explicit purpose of providing him with a set of controls for modifying the input in some way so as to create new audio that blends in and enhances the live improvisation.

The most relevant aspect of Wessel's research is his modeling of music as a stochastic process, using these models for novel generation and/or subsequent modification. Statistical constructs have been often used to operationalize his tools. The Markov chain melodic generational process being developed by Steinsaltz and Wessel (in progress) is a recent example.

8.4.4 George Lewis

Somewhere in between the two different authoring purposes of Dannenberg and Wessel sits the research of George Lewis. Lewis's computer performers are designed to play in the context of improvised music, the point to infuse a *separate, recognizable personality* into its behavior which can participate in musical discourse on an equal footing with humans. While the composer is responsible for some aspects of his systems' behaviors (i.e., setting certain probabilities) other aspects are owned entirely by the systems themselves (i.e., harmonic and rhythmic input does not react to what the musician plays) (Rowe 1993).

Lewis describes his *Voyager* system as quite open-ended, pointing out that the emergent properties that arise "when playing with the thing" are of primary interest to him (Casserly 1997; Roads 1985). In this sense, his motivation and purpose in systems building leans more towards Wessel's (and my) views than Dannenberg's. At the same time, Dannenberg and Lewis, both practicing improvisers, have spent years hand-crafting their interactive software environments and in this sense, Lewis's impact on AI and music is akin to the impact that Harold Cohen has had on AI and drawing.[2]

[2]Cohen created the *Aaron* drawing program (McCorduck 1991), which can generate astonishingly humanlike sketches using algorithms that, in their essence, are a life-long attempt to mechanize Cohen's artistic notions.

8.4.5 Other Related Interactive Systems

Improvisation Builder

Walker (1997) proposed *Improvisation Builder* (IB) in order to break out of the authoring camp. A primary distinction of IB is that it can play jazz with a musician without relying on any sort of lead sheet. Rather, conversation analysis techniques were used to infer what IB should do at a given time (to solo or to accompany). As more sophisticated interaction scenarios are developed, it will make sense to consider integrating these techniques with BoB.

The Reactive Accompanist

The goal of the *reactive accompanist* was to provide real-time chordal accompaniment to a folk melodist playing without a score (Bryson 1992). This research embraced the subsumption architecture (Brooks 1990), in stark contrast to a symbolic, knowledge-based approach. Thus it was natural for the system to operate on live acoustic input. In particular, a neural network was trained on a set of accompaniment examples, learning to predict what chord should be played given a recent window of acoustic input. One aspect of this system that, in lieu of its subsumption basis is surprising, is that it did not handle other important lower-level issues, like what note sequences should realize specific chord sequences. Although BoB does not operate directly on live audio, its most basic representation (the VLT) tackles the other analogous low-level issue, i.e., what notes to play for a specific melodic mode.

GenJam

GenJam is a genetic algorithm-based model of a jazz musician learning to improvise (Biles 1994). The most relevant aspect of this work is that human listeners' aesthetic preferences are used to drive the generation of the next set of improvisations, i.e., this system is a concrete example of how one might automatically convert a listener's musical sensibility into operational procedures. One reason why I did not try a similar approach was that it seemed unreasonable to expect IMC users to listen to and rate many examples. In addition, this data collection procedure necessarily removes the musician from improvisation's most fundamental aspect: its *physical* and spontaneous creation of melody. While the original system was not interactive, later work *GenJam* was enhanced so that a live musician could, for example, trade fours with it (Biles 1998). Attempts to address the system's "fitness bottleneck" were also explored (Biles 1995; Biles 1996; Biles 1998).

CHIME

Recently, Franklin (2001) developed *CHIME*, an agent that learns to trade solos with a jazz musician in two different phases. CHIME represents melody as a sequence of pitches and rests with sixteenth-note durations (slurs were used to create longer notes). In phase one, a recurrent neural network learns how to reproduce three jazz melodies (taken from saxophonist Sonny Rollins' transcriptions). In phase two, a real-valued reinforcement learning algorithm was used in conjunction with a human performer's solos to refine the network's behavior. The reward signal, which Franklin (2001) constructed by hand, rated the agent's solos in terms of simple jazz-based harmonic criteria. This system's relation to BoB was also described on pages 67, 78, 80, and 157.

8.5 Machine Learning

8.5.1 Clustering

Multinomial-based clustering has not received as much attention in machine learning as Gaussian-based clustering has (McLachlan and Basford 1988). When algorithms do cluster Multinomials, they often impose special restrictions on the number of counts they may contain. For example, in *Autoclass* (Hanson, Stutz, and Cheeseman 1991), discrete variables are either present or not (i.e., no more than one count per bin is allowed). In order to use *Autoclass* with variable-sized histograms, one needs to break up the counts within a given histogram and model each independently.[3] Another common approach is to insist that each bin contains at most one count (Meila and Heckerman 1998; McLachlan and Basford 1988). During the initial stage of my research, the clustering algorithms that I found in the literature imposed these types of restrictions. As a result, I developing vMn. Later on, Multinomial clustering received more attention (largely because the web made document clustering a hot

[3] J. Stutz, personal communication, December 1999.

topic). For example, in Nigam, McCallum, Thrun, and Mitchell (1998), variable-size labeled and unlabeled histograms were combined into a Multinomial mixture model for classifying documents. In contrast to vMn, in their model, histogram counts were explicitly normalized because they felt that longer documents should not be more heavily considered. More recently, Multinomial clustering has received more attention in its application to the inference of Markov chains (Thom 2000b; Ramoni, Sebastiani, and Cohen 2000b; Ramoni, Sebastiani, and Cohen 2000a; Sebastiani, Ramoni, and Cohen 2000; Smyth 1999; Cadez, Gaffney, and Smyth 2000).

A valuable aspect of using a probabilistic model is that it allows simulated experimentation. In pure machine learning terms, vMn's application to BoB motives two directions for future research:

1. How much does the assumption that segments are independent impact learning performance? This issue could be explored by simulating histogram datasets using a higher-level sequence generator, which allows the correlation between segments to be controlled and the impact on learning performance to be assessed.

2. What changes when CGL histograms are clustered as an entire unit rather than in separate PC, INT and DIR models? This question makes sense when one considers that various bins in PC, INT and DIR might be correlated with one another. While in the Multinomial, bins are modeled as independent, in terms of classification, they can still collaborate with one another towards improved discrimination.[4]

8.5.2 Sequence Learning

Both Feulner and Hörnel (1994) and Mozer (1994) used neural networks to learn melodic structure. Once trained on input melodies, these networks were able to produce "similar" output melodies. One complaint with these schemes was that their generated melodies lacked structural coherence (Hörnel 1998). An interesting method for dealing with this was proposed in Hörnel (1998): construct networks that operate over multiple time scales. The basic idea is that melodies are comprised of motives, within which distinct note-by-note generation schemes will be operative. By breaking the problem into two separate learning networks, one could expect better predictive power (and hence better generative power) because the consideration of what happens

[4]The essential line of reasoning for why this is true is argued in a different context in Smyth (1999).

next (the next note, the next motive) is now explicitly organized around a hierarchical scheme that is likely to be influencing the actual production of melody. Hörnel (1998) demonstrated impressive results for a system trained in this way on the melodies of Pachelbel. BoB has an analogous set of clusters — preferences for certain tonalities, intervals and contours. A compelling direction for future research is to perform higher-level sequence learning on this kind of data (as was described in Section 7.2.1).

This investigation becomes more interesting when one considers that higher-level sequence prediction is easier when the size of its alphabet (e.g., the number of motive classes) is small because there are less ways in which to guess the next motive incorrectly. Similarly, the smaller the number of classes, the less distinction between melodic segments within a class, which makes the prediction of the next note more difficult. As demonstrated in (Hörnel 1998), there is a complex interaction between these two affects, and one would expect that at the "right" level of granularity an "optimal" model might be obtained. This raises the exciting possibility of unifying into one algorithm both learning to predict higher-level sequences and learning to cluster the lower-level sequences that are transformed into these higher-level classes. Such an algorithm would be of great value in temporal, discrete, unsupervised domains.

In future work, I plan to use a universal compression algorithm (UCA) to learn discrete sequences in BoB. Such an algorithm is ideal because it asymptotically achieves the best possible compression rate for any source distribution (Lartillot, Dubnov, Assayag, and Bejerano 2001). This performance is because the learner can build variable-sized contexts into its prediction algorithm as needed. Clearly, this capability is ideal in musical contexts. UCAs have already been used to produce impressive results when modeling musical structure (Dubnov, Assayag, and El-Yaniv 1998; Assayag, Rueda, Laurson, Agon, and Delerue 1998; Lartillot, Dubnov, Assayag, and Bejerano 2001).

Chapter 9

Conclusion

In this chapter, the long-term vision that drives this research is recapped and the scientific contributions made to the the fields of machine learning, intelligent agents, and interactive music systems are summarized.

9.1 The Vision

In abstract terms, my vision is to develop technologies that allow the computer to begin participating *as a first-class citizen* in creative, artistic, collaborative exchanges. It is of enormous benefit that artistic goals are at least as valuable as purely technical ones. I firmly believe that good art, when transformed into a computational setting, will breed good science, but that the converse is not necessarily true.

In concrete terms, the vision of my research is to interactively improvise with the computer, using it as a tool when practicing alone to capture and experiment with my all-too-transient spontaneous musical ideas. The penultimate goal would be to someday produce an artifact that could:

> ... answer me halfway through my phrase, bringing me to the point where I can actually sing what its going to play next, and then, instead of playing that, it'll play something against it which compliments what I'm singing in my head.[1]

This goal is an inherently interactive one. Although IMC technology might be

[1] The paraphrased musings of drummer Keith Copeland on improvising with jazz pianist Ahmad Jamal (Berliner 1994).

amenable to generating stand-alone melodies, this "brain-in-a-vat" exchange merely detracts from the main point: to interact with, enhance, and leverage off of the spontaneous creativity of the musician.

9.2 Contributions

This thesis offers five contributions to the fields of machine learning, intelligent agents, and interactive music systems.

9.2.1 The Improvisational Music Companion

Chapter 2 introduced and explored the concept of an agent as a spontaneous musical companion. Of potentially great benefit to AI, this exploration *forces* one to tackle the seemingly impossible task of "operationalizing nothing" because:

> ...music is not about anything in particular...the challenge and potential rewards for both AI and the broader domain of cognitive science lie in attempting to reconcile [its] dual aspects: music is humanly universal yet at the same time appears to be a strangely functionless activity. (Cross 1999)

In tackling this issue in BoB, a new agenda for machine learning and musical intelligence emerged. To oversimplify, less effort was invested engineering specific music details so that more time could be spent building a completely flexible, adaptive, and intimate solo-trading model.

9.2.2 The Hierarchical Set of Melodic Views

Chapter 4 introduced a novel melody representation scheme in which solos were segmented per-bar, providing a highly localized view of the melody. Segments were further decomposed into a set of views that explicitly captured salient aspects of their rhythm, tonality, and melodic continuity and contour

A major contribution is the representation's hierarchical, musically meaningful, multifaceted nature. VLTs are used to build PC, INT, and DIR histograms, which in turn are used to build CGL feature vectors, which in turn are mapped into user-specific playing modes, which in turn display certain degrees of musical surprise. While the more abstract views ensure that the agent can see “beyond the solo’s immediate melodic surface,” the highly localized segments provide the flexibility needed to intimately adapt and respond to the musician. Furthermore, this representation’s hierarchical and musically meaningful handling of rhythm, tonality, and melodic continuity and contour is a big part of why, as demonstrated in Chapter 7, BoB’s solo-trading model is easy to understand, interpret, and (when desired) infuse with higher-level domain knowledge.

9.2.3 Learning a User’s Playing Modes

In Chapter 5, an EM-based variable-sized mixture of Multinomials was introduced to cluster a set of histograms. In BoB, this technology is used to learn to perceive a segment’s tonality and melodic continuity and contour trends in a user-specific way.

With respect to machine learning, the main contribution is this model’s ability to learn structure when datasets are relatively sparse. Counts within the dataset become more powerful indicators of model structure because the vMn model explicitly treats each histograms’ counts as having been generated by a particular underlying Multinomial component. The music contributions of this model were demonstrated in Chapter 7. Within the learned components’ parameters and among their perception of the training data, musically appropriate and powerful musician-specific abstractions are embedded. For example, the two most probable PC components learned for Parker’s model have natural Bebop scalar interpretations. This result is impressive because the learner had no explicit tonal/harmonic knowledge and the training set was very small.[2] To fully appreciate this result, one must realize that these scales were defined by musicians after the fact to explain the improvisations that Parker had a hand in inventing.

9.2.4 Transforming a Solo Call into an Appropriate Response

In itself, user-specific perception does not provide a means by which the computer can respond to a musician. Another contribution of this thesis is an algorithm that “closes the learning loop,” reversing the learned mapping from solos to user-specific playing modes. In chapter 6, this algorithm, in conjunction with another that tweaks

[2]Containing only 119 histograms, with an average of 7.5 counts in total distributed among a histogram’s 12 bins.

a call's rhythmic structure, were introduced, providing the functionality needed to generate novel user- and context-specific solo responses.

Reversing the user-specific model was accomplished by incorporating the desired component's learned parameters into a Markov chain that, when sampled, would produce pitch sequences that tend to exhibit the desired conglomeration of trends. Another important aspect of this algorithm is its ability to simultaneously ensure that a fairly arbitrary set of constraints are maintained, providing a natural mechanism for including rhythmic, contextual, and/or domain specific considerations later on. Another crucial aspect concerns real-time efficiency, which was feasible only when some independence assumptions were made about how to incorporate the learned mode's parameters into the generator.

The main contribution of the tree-tweaking algorithm is its hierarchical nature, providing an ideal basis for reasoning about rhythmic modification. Similarly, the main contribution of the pitch sequence generator is the ideal basis it provides for reasoning about a melody's pitch content. As a result, the algorithm is quite musically powerful. To cite one example, straight out-of-the-box, this algorithm could be used to generate a pitch sequence that starts and ends on specific tones and "fills in the blanks" in such a way that a particular conglomeration of trends emerges. This approach was demonstrated to successfully reverse learned mappings and maintain constraints in a variety of realistic musical settings. The listening experiments presented in Chapter 7 also indicate that controlling generation on a per-user and per-mode basis has noticeable impact.

9.2.5 The BoB Solo-Trading Architecture

Chapter 3 introduced an overall architecture where this thesis' representation, learning, perception, and generation components were combined to produce BoB, an agent that trades user- and context-specific solos with a musician in real time (albeit presently via simulation). An important aspect of this architecture is that the user-model is learned offline (automatically configuring itself to about 15 minutes of warmup data). While real-time online modification of this model is a feasible possibility, it is unlikely to expect that offline learning can be eliminated altogether.[3] Online, it is trivial to perceive user playing modes in real-time. With minimal effort (or a

[3]Depending on how careful one wants to be in learning the number of components in a user-model, offline learning can take anywhere from a few minutes to a few days.

slightly faster computer) the generation algorithm can execute in real-time (currently about 3 seconds per bar are needed).

9.3 Closing Remarks

Before closing, a few remarks about the call-and-response metaphor that was used to ground human-computer interaction are in order. It is easy to make an argument that call-and-response is in essence a toy problem, and taken literally, I agree with this statement. However, this task has only been used to situate BoB, and from within this paradigm, much more cooperative modes of interaction can be imagined. For example, scheduling issues aside, what the computer needs to be able to move beyond call-and-response is the ability to predict *what is coming next.* This skill requires a user-model that can:

- perceive strings-of-notes with some degree of abstraction;
- generate abstract intentions on a note-by-note basis.

Band-OUT-of-a-Box presents a working system that begins to tackle both of these issues. With such a model in hand, meaningful and interesting applications can follow.

Appendix A

Interpreting Pseudo-code

All of the algorithms presented in this thesis are outlined in pseudo-code, presented with the goal of supporting reimplementation. Pseudo-code is presented in tables (e.g., Table 5.4), the main body enclosed in a large box, and the function's name and argument enclosed again in another. When individual lines need to be identified, they begin with a number followed by a colon. This number serves as the line's tag. For readability, some algorithms are broken up into a set of functions.

The `Terminal` type font is used to identify functions as well as primitive programming keywords (e.g., `for`, `Argmax`, `abs`, etc.). Functions are defined using pseudo-code that loosely follows *C++*-like conventions. For example:

- Operator '++' increments a variable by one.
- Return values are passed using the `return` keyword.
- Multiple assignments per line are separated by a ';'.
- Function application takes place when the function is followed by an argument list, '$(\cdots)$'.
- When the reference operator, '&', is used, it indicates that an argument is being passed by reference.

In the main body of a function, mathematical variables follow the same conventions that hold for formulas. As a result, in the following context, if a tuple is returned by a particular line in function `blah`:

$$\texttt{return}\langle foo, bar\rangle,$$

and it is assigned via

$$bash = \texttt{blah}(),$$

then *bash*, and *foo* and *bar*, might be used interchangeably (and vice versa).

For tree-based functions, an object oriented style of pseudo-code is used. Again, syntax loosely follows *C++*-like conventions. For example:

- class methods are declared with the '$<$ `class` $>$::$<$ `method` $>$' identifier.
- class methods and fields are retrieved from objects using the '.' operator, and when inside a given method, these fields are accessed using keyword '`this`'.
- the name of a class's constructor has the same name as its class.

A major simplification is that pseudo-code glosses over low-level memory related issues and garbage collection is assumed to automatically take care of itself. As a result, access to an object (or `this`) always uses the '.' operator. Similarly, constructors are executed as regular function calls (as opposed to using `new`). When these constructors receive arguments, it will be clear from context what fields within the object these are used to initialize.

Appendix B

Interpreting Music Scores

Musical scores adhere to the following conventions:

1. Bar marker '|' identifies where one bar ends and the next begins.

2. Bars are numbered and displayed in the order in which they were played. A small number at the upper left-hand column of a staff refers to the number of the bar that begins that staff.

3. During interaction, two staves are used, one for the call's bars and one for the response's.

4. When text to the left of a staff appears, it indicates who played that staff's notes ('Parker' or 'P' for Parker; 'Grappelli' or 'G' for Grappelli; 'Bob' or 'B' for BoB).

5. Harmonic accompaniment chords appear underneath the a stave. Standard chord notation is used.

6. Section marker '||' serves to break up choruses and/or separate calls and responses.

7. When three comma-separated numbers are printed in **bold** type above and to the left of a bar, it is the bar's *conglomerative class tag.* A bar's tag indicates to which user-playing mode that a segment has been assigned. For example, if a bar was tagged with **1,3,2**, then a model's first PC component, third INT component, and second DIR component are most likely to have generated it.

The *Lilypond* program was used to produce BoB's musical scores (Nienhuys, Nieuwenhuizen, and Mariano 1999). When BoB converts its tree-based melodic representation (Section 4.3) into a Lilypond score, no effort is made to transform it into a more succinct Lilypond input. This decision impacts the scores produced in several ways:

1. Since syncopated rhythms are represented via compound durations, rhythms are not always printed in their "simplest" form (e.g., bars 50, 109, and 110 in Parker's training data).

2. Since leaves do not contain articulation information, slurs are only drawn when rhythms are syncopated.

3. Pitches are notated without reference to a musical key. Chromatic tones are notated using ♭-based notation rather than a #-based equivalent, even when, diatonically speaking, it is less appropriate to do so.

Although not standard in music notation practice, when *Lilypond* displays a bar whose first note is slurred with a previous bar's identical, but chromatically altered tone, it assumes that this alteration holds during the entire bar. Thus, for example, in bar 110 of Parker's training data (page 223), all of the first sixth eight notes are B♭.

Appendix C

VLT Construction

This pseudo-code accompanies Section 4.3. For details concerning pseudo-code notation, see Appendix A.

The `makeVLT` function (Table C.1) parses a list of notes, s, attempting to convert its corresponding list of transcribed durations into a sequence of valid durations. If a sequence of valid durations exists, then the VLT is completely specified, the function returns a status of `true`, and the corresponding VLT is stored in v. Equivalently, this algorithm returns `false`, failing to build v, when the transcription specified by note list s is not encodable. Note object *curr* is the current event in the transcription that is attempting to be converted into a valid duration (or set of valid durations). The valid duration, d, that is currently being considered, is sought for by recursively calling `makeVLT` on smaller valid durations until one that can encode at least part of *curr* is found.

This code relies on the `leaf`, `int2`, and `int3` classes (Section 6.2.1). A note class is also assumed, which stores duration, pitch, and tie information. In addition, it is assumed that mathematical operations on fractional durations are handled with perfect precision and that a 4/4 meter and per-bar segmentation scheme is used.

`makeVLT` is a recursive search algorithm, parsing s from left to right, each call attempting to use part or all of *curr*'s duration to construct the next node in v. What this algorithm recurses over is the duration of the current internal VLT node, d, being considered (each recursion follows one of links shown in Figure 4.2).

When `doSomething` returns `true`, it means that another node in v was constructed by either:

$\boxed{\texttt{makeVLT}(\&curr, \&s, \&v, d)}$

1: **if** isDone($s, curr$)
  return false
2: **elseif** !doSomething($s, curr, d, v$)
3:   $s_{halve} = s$; $curr_{halve} = curr$; $v_{halveLeft} = v_{halveRight}$ = an empty tree
4:   ok_{halve} = isValid($1, 2, d$) **and** makeVLT($curr_{halve}, s_{halve}, v_{halveLeft}, \frac{d}{2}$) **and**
    makeVLT($curr_{halve}, s_{halve}, v_{halveRight}, \frac{d}{2}$)
5:   $s_{third} = s$; $curr_{third} = curr$; $v_{thirdLeft} = v_{thirdMid} = v_{thirdRight}$ = an empty tree
6:   ok_{third} = isValid($1, 3, d$) **and** makeVLT($curr_{third}, s_{third}, v_{thirdLeft}, \frac{d}{3}$) **and**
    makeVLT($curr_{third}, s_{third}, v_{thirdMid}, \frac{d}{3}$) **and** makeVLT($curr_{third}, s_{third}, v_{thirdRight}, \frac{d}{3}$)
  if ok_{halve} **and** ok_{third} **and**
7:    total number of nodes in $v_{halveLeft}$ and $v_{halveRight}$ $\leq$
     total number of nodes in $v_{thirdLeft}$, $v_{thirdMid}$ and $v_{thirdRight}$
      ok_{third} = **false**
  end
  if ok_{halve}
8:    $s = s_{halve}$; $curr = curr_{halve}$; v = int2($v_{halveLeft}, v_{halveRight}$)
    return true
  elseif ok_{third}
9:    $s = s_{third}$; $curr = curr_{third}$; v = int3($v_{thirdLeft}, v_{thirdMid}, v_{thirdRight}$)
    return true
  else
    return false
  end
else
10: **return true**
end

$\boxed{\texttt{isDone}(s, curr)}$

return s is empty **and** $curr$.dur $== 0$

$\boxed{\texttt{doSomething}(\&s, \&curr, \&v, d)}$

if $curr$.dur $== d$
11: v = leaf($curr$); $curr$ = pop off first element in s
  return true
elseif $curr$.dur $> d$
12: v = leaf($curr$.pitch, d, $curr$.tie); $curr$ = note($curr$.pitch, $curr$.dur $-$ d, true)
  return true
else
  return false
end

Table C.1: The makeVLT Encoable Sequence to VLT Converter

1. using up all of *curr*'s duration, in which case the next element of the list is retrieved (line 11);

2. using up part of *curr*'s duration, in which case the amount that remains to be placed in v is reduced (line 12).[1]

In either case, work has been done and `makeVLT` exits its current recursion with a successful status (line 10).

When work is not immediately possible (the block beginning at line 2), the search for a shorter valid note duration begins, recursively attempting to subdivide by both two (line 4) and three (line 6). Both splits must be attempted in order to ensure that Restriction III can be enforced via the *size check* at line 7. Note that either of these splits (or their subsequent `makeVLT` calls) could fail, in which case the search backtracks to the next viable possibility.

If line 1 is ever called, it means that all of s has been placed in v but more children are required, hence an unsuccessful status is returned.

The reason that `makeVLT` only terminates with a `false` status when s is not encodable is because, in the worst-case, it tries every possible combination of subdivisions before giving up. Backtracking, which undoes the unfruitful recursions that do not pan out (lines 3, 5, 8, and 9), is the main reason why `makeVLT` is complex. When `makeVLT` returns with a `true` status, v is guaranteed to be a VLT because:

1. All subdivisions have been recursively tried.

2. `isValid` ensures that Restrictions I and II are maintained.

3. The size check ensures that Restriction III is maintained.[2]

[1] It is this line of code that ultimately produces compound durations (Section 4.3.4). The key point is that the last field of the note object, which contains the value of its tie, remains `true` until the compound duration is completely inserted into v.

[2] This statement assumes that a local size check can produce the globally smallest tree.

Appendix D

Learning Appendix

This appendix accompanies Chapter 5.

D.1 Multinomial Sampling

This pseudo-code accompanies Section 5.7.1.

$\boxed{\texttt{MnomSample}(sz, \theta)}$

for $\mathsf{m} = 1;\ \mathsf{m} \le \mathsf{M};\ \mathsf{m}++$
  if $\mathsf{m} == 1$
    $x_{\mathsf{m}} \sim \mathrm{Binom}(sz, \theta_{\mathsf{m}})$
  elseif $\mathsf{m} < \mathsf{M}$
    $$x_{\mathsf{m}} \sim \mathrm{Binom}\left(sz - \sum_{\mathsf{m}'=1}^{\mathsf{m}-1} x_{\mathsf{m}'}, \frac{\theta_{\mathsf{m}}}{\sum_{\mathsf{m}'=\mathsf{m}}^{\mathsf{M}} \theta_{\mathsf{m}'}}\right)$$
  else
    $$x_{\mathsf{m}} = sz - \sum_{\mathsf{m}'=1}^{\mathsf{M}-1} x_{\mathsf{m}'}$$
  end
end
return x

D.2 Multinomial Versus Gaussian Components

A Mnom's mean, variance, and covariance (Section 5.7.1) can be used to define a Gaussian distribution. When sz is large enough, this definition provides a good approximation to the Mnom distribution. The reason I originally considered such an approximation scheme for clustering histograms is because Gaussian mixture model implementations are so common. However, I abandoned this idea for two reasons. First, the Gaussian approximation becomes less appropriate as sz decreases. Second, the histogram clustering model should explicitly consider the number of counts.

D.2.1 The Guassian Multinomial Approximation

In Figure D.1, different 2-dimensional Mnom distributions are displayed in order to provide an indication of how seriously sz affects the quality of the Gaussian approximation when counts are as sparse as they are in BoB's representative datasets.

Each plot is a different distribution. The distributions are arranged as follows:

1. each row of distributions uses the same sz value, ranging from one count (top row) to thirty (bottom row).

2. each column of distributions uses the same probability vector, the left uniformly distributed, $\theta = \langle 0.5, 0.5 \rangle$, and the right uneven, $\theta = \langle 0.75, 0.25 \rangle$.

The Multinomial probability distribution is drawn as a solid line, and the Gaussian approximation is dashed. A single bin is sufficient for displaying 2-dimensional distributions because, since sz is known, the other bin is completely specified. An important subtlety is that Mnom is only defined on integer values (marked by asterisks). The error between an Mnom and its Gaussian approximation is worse when θ is less uniform.

D.2.2 Considering Counts

The Gaussian approximation to the Mnom assumes that sz is fixed. One way to accomodate this assumption would be to normalize the histograms.[1] Clustering could then be performed on the probability vectors that result. In BoB, however, it makes sense to consider those histograms with more counts more heavily. Typically, when a musician plays a very fast passage of notes, it is easier to identify an underlying scale because there is more information available upon which to base ones assessment.

[1] Histogram normalization is often done in text learning, e.g., (Nigam, McCallum, Thrun, and Mitchell 1998).

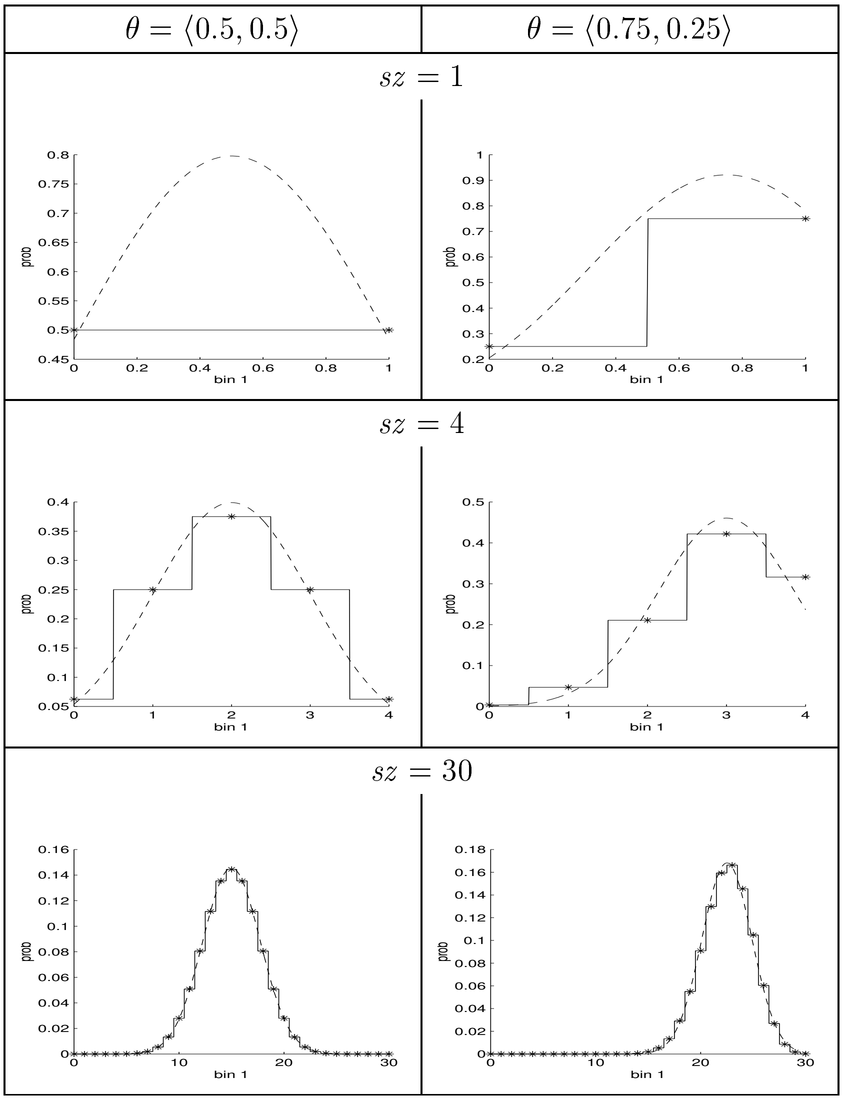


Figure D.1: The Multinomial and Its Gaussian Approximation

D.3 Quality Assessment

This pseudo-code accompanies Section 5.7.5.

D.3.1 Comparing Solution Partitions

`diff` compares two specific dataset partitions. Only when one or both of these clusterings is based on learning does consistancy relabeling enter into the comparison. This logic is handled by the *doMap* variable.

```
diff(Y, Y', doMap)
    if doMap
      remap Y' so that its labels are as consistent with Y as possible
    end
    numDiff = 0
    for n = 1; n ≤ N; n++
      if y_n ≠ y'_n
        numDiff++
      end
    end
    fracDiff = numDiff / N
    return fracDiff
```

D.3.2 estHardness

```
estHardness(Ω)
    ⟨X, Y⟩ ~(n_sim) vMn(Ω)
    Ŷ = classifyVmn(X, Ω)
    hardness = diff(Y, Ŷ, false)
    return ⟨hardness, X, Y⟩
```

D.3.3 estCost

$\boxed{\texttt{estCost}(\Omega)}$

$X^{\boldsymbol{Tr}} \overset{n_{sim}}{\sim} \mathrm{vMn}(\Omega)$

$\Omega^{\boldsymbol{Tr}} = \texttt{learnVmn}\Omega(X^{\boldsymbol{Tr}}, \mathsf{K})$

$\langle hardness, X^{\boldsymbol{Te}}, \underline{Y}^{\boldsymbol{Te}} \rangle = \texttt{estHardness}(\Omega)$

$Y^{\boldsymbol{Te}} = \texttt{classifyVmn}(X^{\boldsymbol{Te}}, \Omega^{\boldsymbol{Tr}})$

$cost = hardness - \texttt{diff}(\underline{Y}^{\boldsymbol{Te}}, Y^{\boldsymbol{Te}}, \texttt{true})$

$\texttt{return}\ cost$

D.3.4 estLearn

$\boxed{\texttt{estLearn}(\Omega)}$

$\langle X^{\boldsymbol{Tr}}, \underline{Y}^{\boldsymbol{Tr}} \rangle \overset{n_{sim}}{\sim} \mathrm{vMn}(\Omega)$

$\Omega^{\boldsymbol{Tr}} = \texttt{learnVmn}\Omega(X^{\boldsymbol{Tr}}, \mathsf{K})$

$overFit = \mathrm{U}(X^{\boldsymbol{Tr}}|\Omega)$ - $\mathrm{U}(X^{\boldsymbol{Tr}}|\Omega^{\boldsymbol{Tr}})$

$\langle X^{\boldsymbol{Te}}, \underline{Y}^{\boldsymbol{Te}} \rangle \overset{n_{sim}}{\sim} \mathrm{vMn}(\Omega)$

$underGeneralize = \mathrm{U}(X^{\boldsymbol{Te}}|\Omega)$ - $\mathrm{U}(X^{\boldsymbol{Te}}|\Omega^{\boldsymbol{Tr}})$

$\texttt{return}\ \langle overFit, underGeneralize \rangle$

D.3.5 obsInstabilities

$\boxed{\texttt{obsInstabilities}(X, \mathsf{K})}$

$\texttt{for}\ \mathsf{n} = 1;\ \mathsf{n} \leq n_{expt};\ \mathsf{n}{+}{+}$

$\quad \Omega = \texttt{learnVmn}\Omega(X, \mathsf{K})$

$\quad Y_{\mathsf{n}} = \texttt{classifyVmn}(X, \Omega)$

$\texttt{end}$

$\mathsf{n} = 1$

$\texttt{for}\ \mathsf{n}' = 1;\ \mathsf{n}' \leq n_{expt};\ \mathsf{n}'{+}{+}$

$\quad \texttt{for}\ \mathsf{n}'' = \mathsf{n}' + 1;\ \mathsf{n}'' \leq n_{expt};\ \mathsf{n}''{+}{+}$

$\qquad instabilities_{\mathsf{n}} = \texttt{diff}(Y_{\mathsf{n}'}, Y_{\mathsf{n}''}, \texttt{true})$

$\qquad \mathsf{n}{+}{+}$

$\quad \texttt{end}$

$\texttt{end}$

$\texttt{return}\ instabilities$

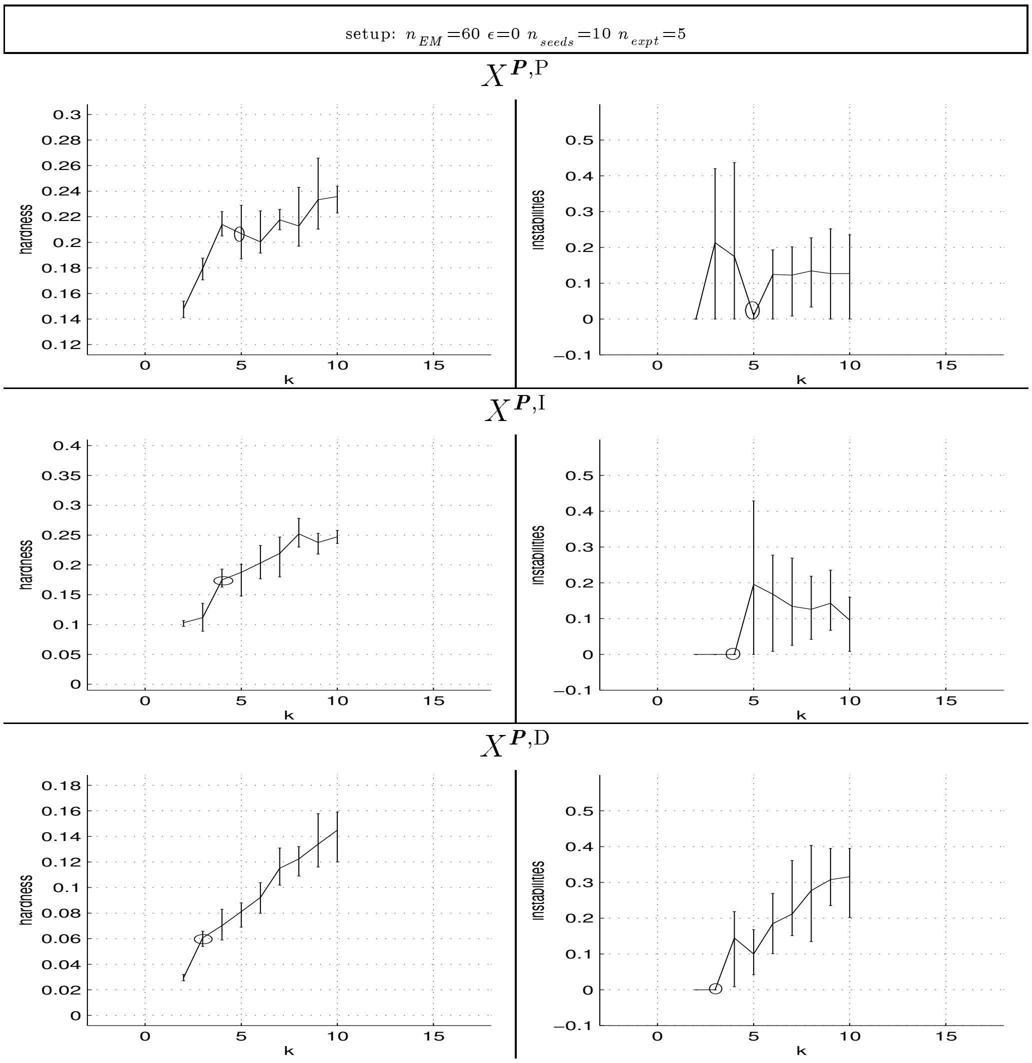
D.3.6 `obsFitnesses`

```
obsFitnesses(X, K)
  for n = 1; n ≤ n_expt; n++
    ⟨fits_n, Ω⟩ = learnVmnΩ(X, K)
  end
  n = 1
  for n′ = 1; n′ ≤ n_expt; n′++
    for n″ = n′ + 1; n″ ≤ n_expt; n″++
      fitDiffs_n = fits_n′ − fits_n″
      n++
    end
  end
  translate fitDiffs vector so that its minimum value is 0
  return ⟨fits, fitDiffs⟩
```

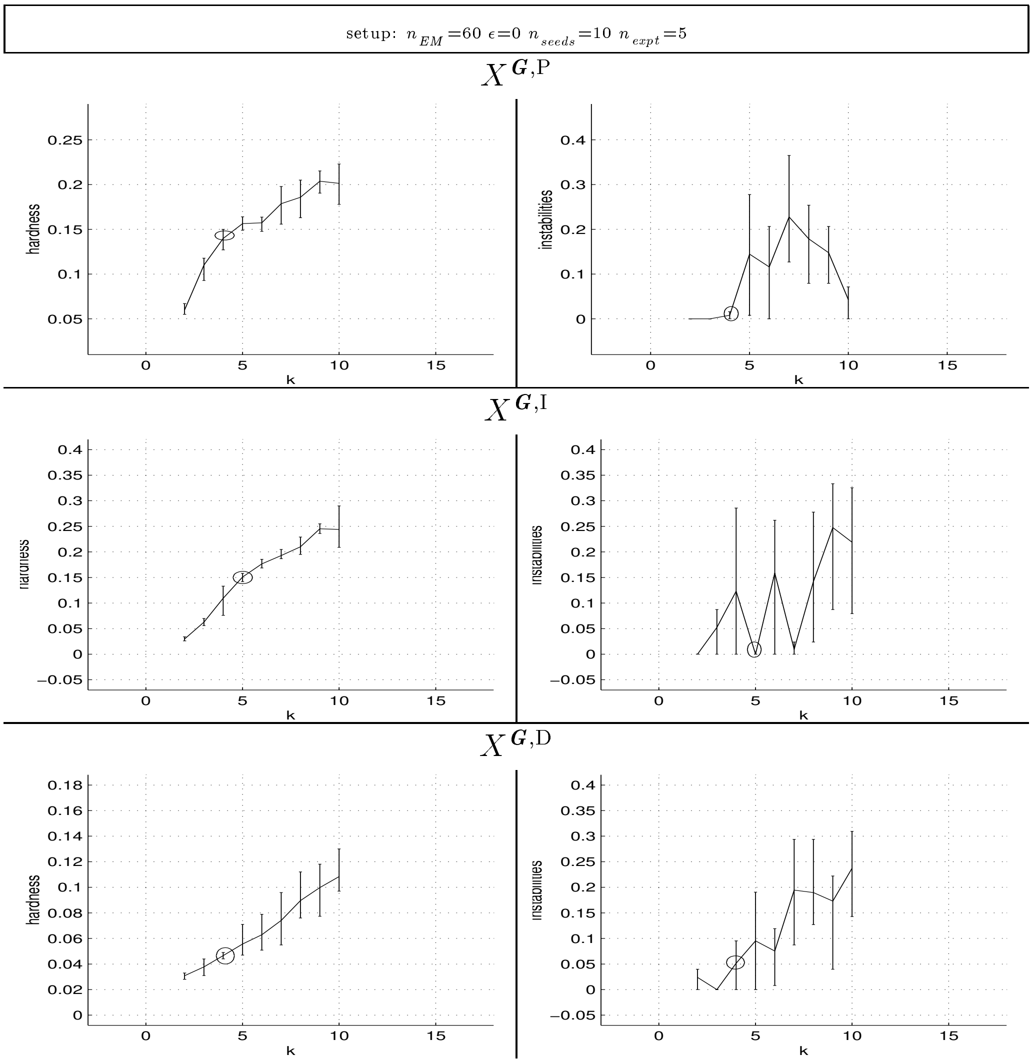
D.4 Musician-Specific Hardness And Instabilities Curves

These figures accompany Section 5.8.1. For each learning task, the chosen "best" number of components (Table 5.8) is enclosed in a small circle.

D.4.1 Charlie Parker



D.4.2 Stephan Grappelli



Appendix E

The Charlie Parker Model

This appendix, which accompanies the customized perception of Charlie Parker (Section 7.2), lists the improvisations used to train Parker's model, shows how the model that was learned segregates these improvisations, and displays graphs that demonstrate how these models perceive playing modes as a function of underlying harmonic context.

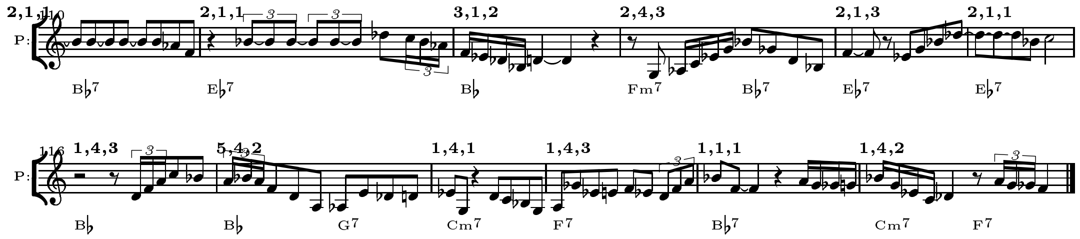
E.1 Training Data

Training dataset $X^{\boldsymbol{P}}$ was created using the music score presented below. This score was obtained as described in Section 3.2.2. Section markers divide the transcription into ten 12-bar choruses. All segments except the pickup (bar 1) and the segment containing no notes (bar 57) were used to construct $X^{\boldsymbol{P}}$, producing $\mathsf{N}^{\boldsymbol{P}} = 119$ datapoints in total.

Each bar's conglomerative class tag is defined according to the learned models in Tables 7.1 through 7.3. These tags were assigned *after* learning took place and were assigned to all the bars in the score, regardless of whether or not they were included in $X^{\boldsymbol{P}}$. As a result, because bar 57's histogram contains no counts, its assignment is based solely on the components' mixture weights.

Mohawk I and *II*


59 1,4,3 5,4,3 1,2,2
F7 B♭7 Cm7 F7
1,2,2 3,1,1 2,4,2 4,1,1 2,1,1 1,3,1
B♭7 E♭7 B♭ Fm7 B♭7 E♭7 E♭7
68 1,3,1 3,4,3 2,1,1 2,1,3 4,1,3
B♭ B♭ G7 Cm7 F7 B♭7
73 4,1,3 2,1,1 2,1,1 4,1,3 2,2,3
Cm7 F7 B♭7 E♭7 B♭ Fm7 B♭7
78 2,1,2 1,3,1 3,1,1 5,4,3
E♭7 E♭7 B♭ B♭ G7
82 1,2,2 1,4,2 2,4,3 4,1,1
Cm7 F7 B♭7 Cm7 F7
2,2,1 2,1,2 4,4,2 5,2,1 2,2,3
B♭7 E♭7 B♭ Fm7 B♭7 E♭7
91 2,2,2 1,4,2 5,4,2
E♭7 B♭ B♭ G7
94 1,4,3 5,3,3 3,1,1 4,1,1 2,4,3 1,2,3
Cm7 F7 B♭7 Cm7 F7 B♭7 E♭7
100 2,1,1 4,1,1 2,2,2 2,1,3 4,1,1
B♭ Fm7 B♭7 E♭7 E♭7 B♭
105 4,2,2 1,1,3 1,2,2 1,1,3 2,1,1
B♭ G7 Cm7 F7 B♭7 Cm7 F7



E.2 Segregated Views

In the following subsections, three segregated views of Parker's training data are presented. All the bars assigned to PC component one (as defined in Table 7.1) are presented first, followed by components two through five, followed by the INT and DIR component models. In other words, the score from Appendix E.1 is rearranged and presented in three different ways, according to the learned tonal, melodic continuity, and melodic contour based partitions. Because INT and DIR assignment relies on a bar's MRP, in these partitions, each bar's pickup note is also included.

Per subsection, components are further ordered (from left to right, top to bottom) according to their CGL mapping. For example, in Section E.2.1, the two bars assigned to conglomerative class $y^{\mathrm{C}} = \langle 1, 1, 1 \rangle$ (i.e., conglomerative class tag '1, 1, 1') are listed first, followed by the four bars assigned to $y^{\mathrm{C}} = \langle 1, 1, 3 \rangle$, and so on. That, for example, $y^{\mathrm{C}} = \langle 1, 4, 1 \rangle$ appears in the list in isolation indicates that it was the only bar in Parker's transcription that was assigned to that conglomerative class. While some CGL classes account for only a single training bar (and other classes, like $y^{\boldsymbol{P},\mathrm{C}} = \langle 1, 2, 1 \rangle$, are not shown at all because they do not account for any bars), separately, each PC, INT, and DIR component accounts for all the bars in a given section.

E.2.1 PC Component 1

E.2.2 PC Component 2

E.2.3 PC Component 3

E.2.4 PC Component 4

E.2.5 PC Component 5

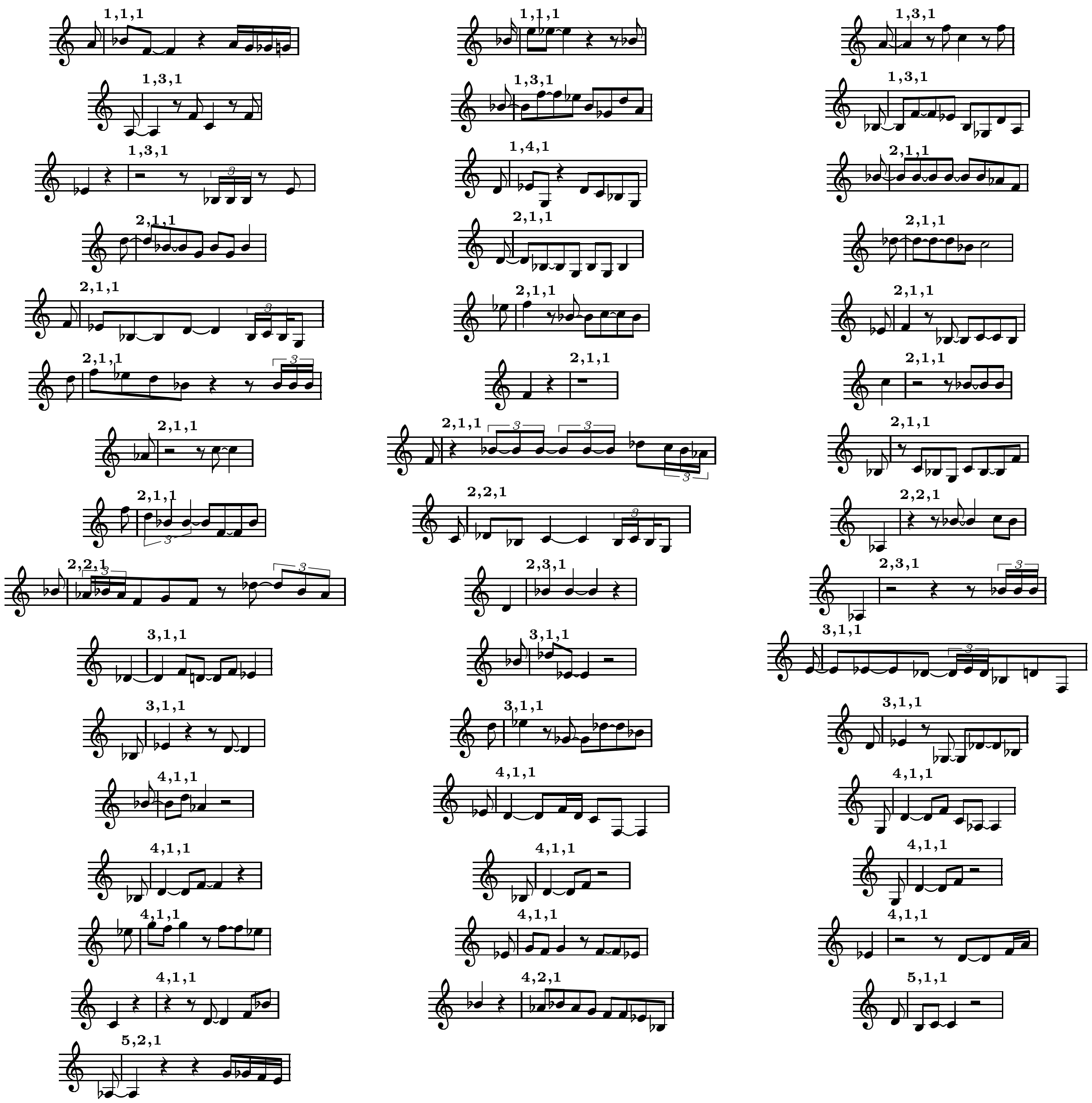
E.2.6 INT Component 1

E.2.7 INT Component 2

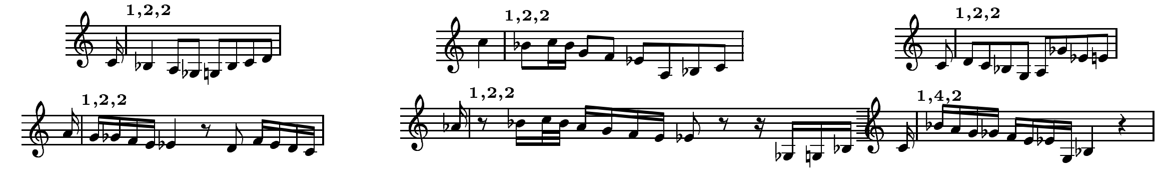
E.2.8 INT Component 3

E.2.9 INT Component 4

E.2.10 DIR Component 1



E.2.11 DIR Component 2



E.2.12 DIR Component 3

E.3 Melodic Continuity in Time

Figure E.1 presents an analogous view for INT component usage that Figure 7.1 did for PC. Each symbol corresponds to one of the bars in Figure 3.1 and identifies which component in Table 7.2 that BoB deemed most likely to have generated that bar's intervallic content.

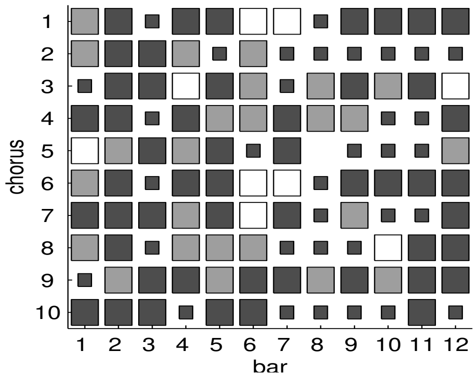


Figure E.1: Parker's INT Modes and the 12-bar Blues

E.4 Melodic Contour and Harmony

Figure E.2 presents an analogous view for DIR component usage that Figure 7.1 did for PC. Each symbol corresponds to one of the bars in Figure 3.1 and identifies which component in Table 7.3 that BoB deemed most likely to have generated that bar's directional content.

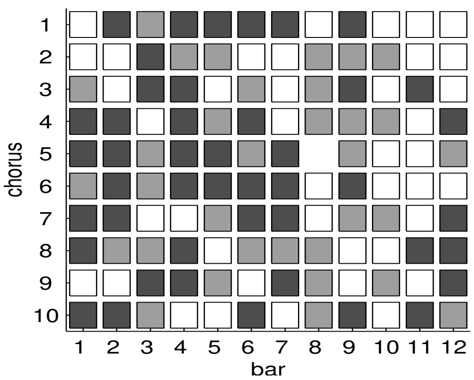


Figure E.2: Parker's DIR Modes and the 12-bar Blues

Appendix F

Learned Grappelli Model

This appendix, which accompanies the customized perception of Stephan Grappelli (Section 7.3), lists the improvisations used to train Grappelli's model, outlines the parameters that the model learned, shows how the model segregates the training improvisations, and displays graphs that demonstrate how these models perceive playing modes as a function of underlying harmonic context.

F.1 Training Data

Training dataset $X^{\boldsymbol{G}}$ was created using the music score presented below. This score was obtained as described in Section 3.2.2. Section markers divide the transcription into four 32-bar choruses. All segments except the pickup (bar 1) and the segments containing single notes (bars 40 and 74) were used to construct $X^{\boldsymbol{G}}$, producing $\mathsf{N}^{\boldsymbol{G}} = 126$ data-points in total.

Each bar's conglomerative class tag is defined with respect to the learned models in Tables F.1 through F.3. These tags were assigned *after* learning took place and were assigned to all the bars in the score, regardless of whether or not they were included in $X^{\boldsymbol{G}}$. As a result, because the DIR histograms in bars 40 and 74 contain no counts, their DIR assignments were based solely on that component whose mixture weight was largest.

I've Found a New Baby I


G:
F A A D D
G G C7 A7 D D
D D G7 C7 F
A7 D D D D
G7 C7 F A7 D
D D D G7 C7 F
F A A D D G
G C7 A7 D D
D D G7 C7 F A7

F.2 Learned Tonality

The four pitch-class components learned for $X^{\boldsymbol{G},\mathrm{P}}$ are shown in Table F.1. Organization of this table is analogous to Table 7.1. For a listing of how this model partitioned each of Grappelli's transcribed bars, see Appendix F.5.1 through F.5.4.

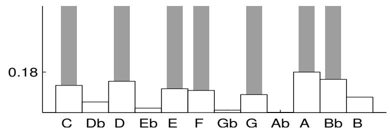
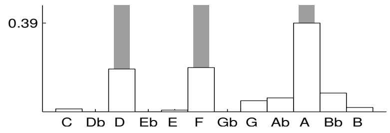
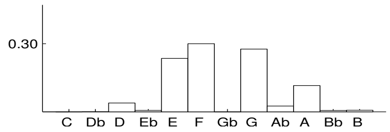
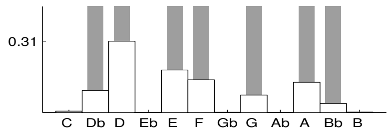
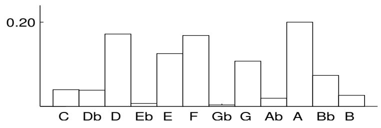
Symbol		Pitch-Class Probabilities	Size		π	Associated Scale
	1		33	26.2%	.292	F Major scale
	2		35	27.8%	.253	D Minor triad
	3		24	19.0%	.185	fragment of the F Major scale
	4		34	27.0%	.270	D Harmonic Minor scale
			a single mixture (for comparison)			

Table F.1: Grappelli's Learned PC Components

F.3 Learned Melodic Continuity

The five intervallic components learned for $X^{\boldsymbol{G},\mathrm{I}}$ are shown in Table F.2. Organization of this table is analogous to Table 7.2. For a listing of how this model partitioned each of Grappelli's transcribed bars, see Appendix F.5.5 through F.5.9.

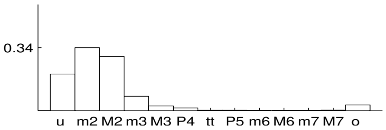
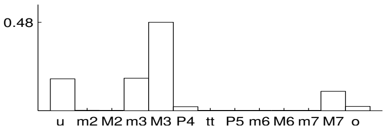
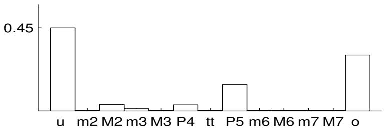
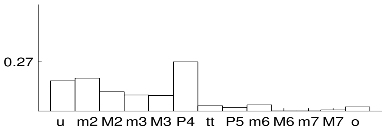
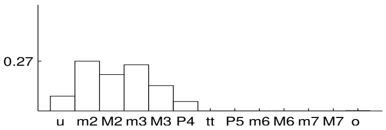
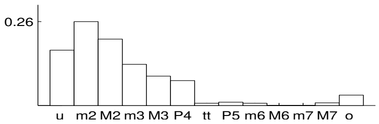
Symbol		Intervallic Probabilities	Size		π	Meaning
	1		60	47.6%	.438	relatively continuous melodies, syncopation and repeated tones
	2		7	5.6%	0.054	somewhat continuous, occasionally jagged or syncopated melodies
	3		6	4.8%	.049	mostly jagged melodies, syncopated or repeated tones
	4		24	19.0%	.213	relatively irregular, syncopated melodies
	5		29	23.0%	.246	very continuous melodies, rarely syncopation or a repeated tone
			a single mixture (for comparison)			

Table F.2: Grappelli's Learned INT Components

F.4 Learned Melodic Contour

The four directional components learned for $X^{\boldsymbol{G},\mathrm{D}}$ are shown in Table F.3. Organization of this table is analogous to Table 7.3. For a listing of how this model partitioned each of Grappelli's transcribed bars, see Appendix F.5.10 through F.5.13.

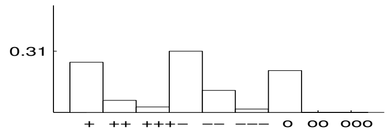
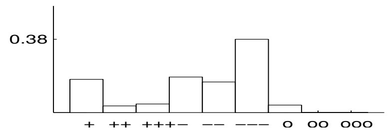
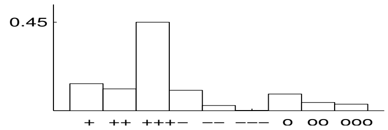
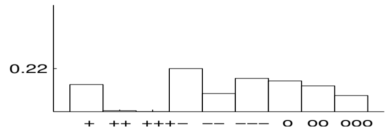
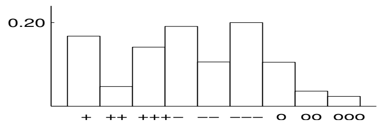
Symbol		Directional Probabilties	Size		π	Meaning
	1		32	25.4%	.236	very syncopated/repeated tone melodies, rarely an obvious direction trend, but for an occassional downards/upwards string
	2		47	37.3%	.372	mostly longer downwards strings separated by upwards steps
	3		22	17.5%	.173	mostly longer upwards strings separated by syncopation, repeated tones, or a short downwards string
	4		25	19.8%	.219	relatively long downards strings separated by syncopation/repeated tones
			a single mixture (for comparsion)			

Table F.3: Grappelli's Learned DIR Components

F.5 Segregated Views

In the following subsections, three segregated views of Grappelli's training data are presented. With the exception that now conglomerative class tags refer to Tables F.1 through F.3, the organization of this section is analogous to Appendix E.2.

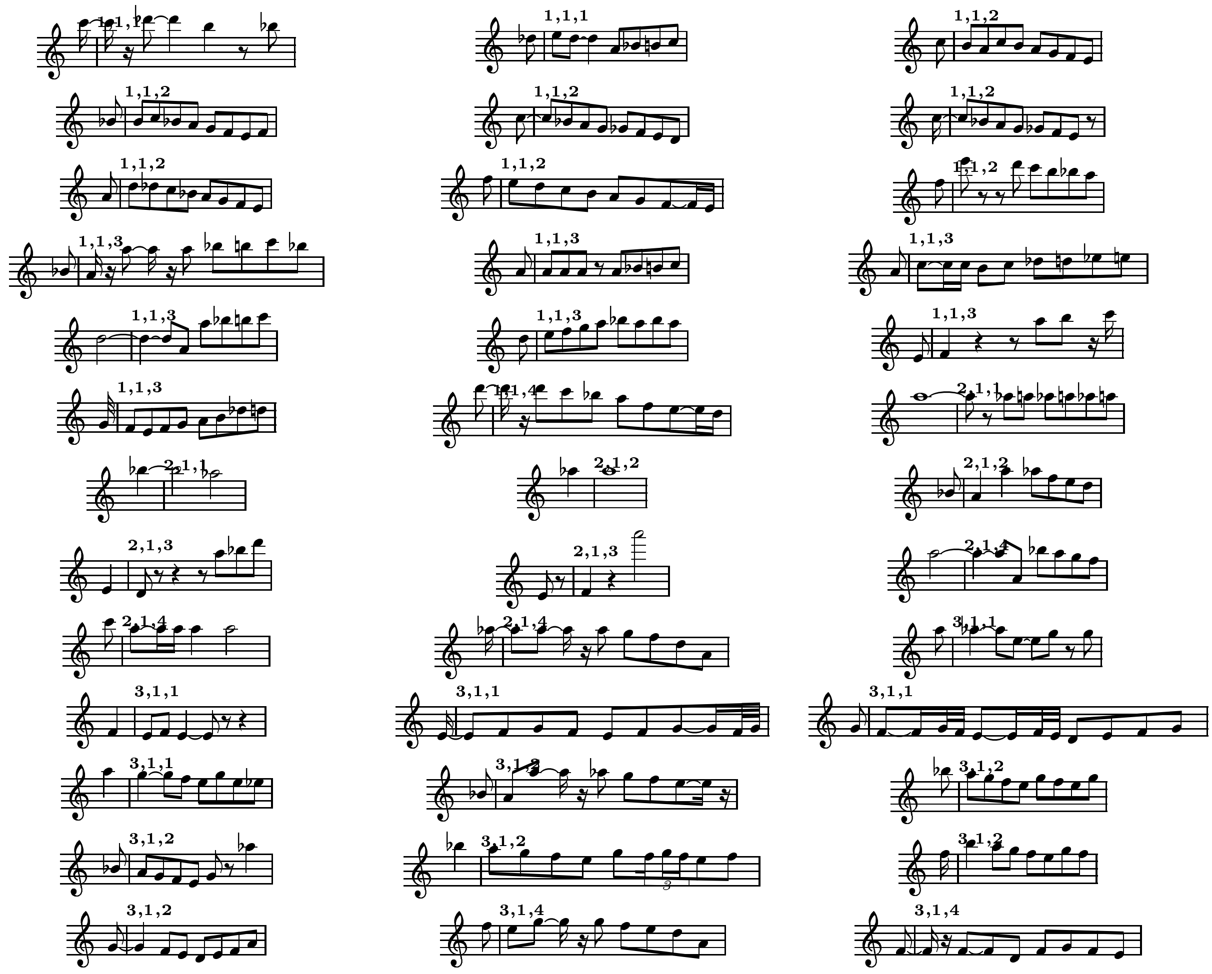
F.5.1 PC Component 1

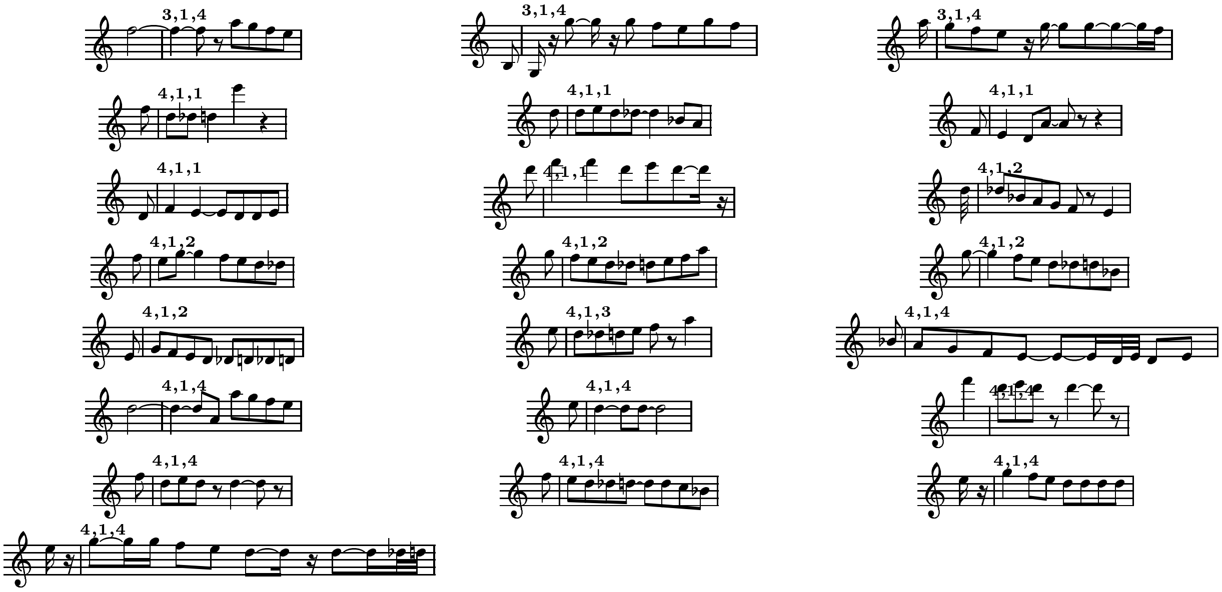
F.5.2 PC Component 2

F.5.3 PC Component 3

F.5.4 PC Component 4

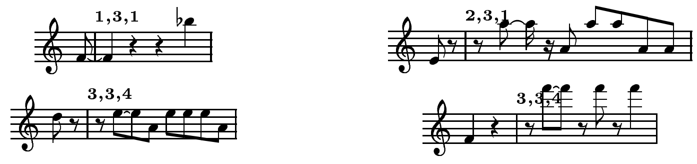
F.5.5 INT Component 1



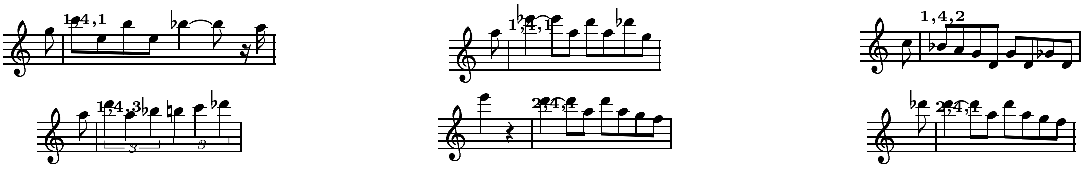


F.5.6 INT Component 2

F.5.7 INT Component 3



F.5.8 INT Component 4



F.5.9 INT Component 5

F.5.10 DIR Component 1

F.5.11 DIR Component 2

F.5.12 DIR Component 3

F.5.13 DIR Component 4

F.6 Tonality in Time

Figure F.1 presents an analogous view for Grappelli's PC component usage that Figure 7.1 did for Parker's. However, in this case, each symbol corresponds to one of the bars in Grappelli's four 32-bar AABA choruses rather than to one of the bars in Figure 3.1. Each symbol is defined in Table F.1.

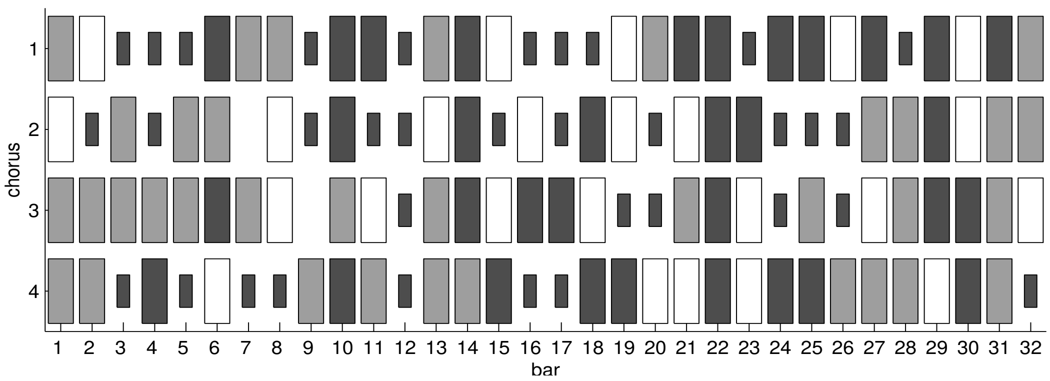


Figure F.1: Grappelli's PC Modes and the 32-bar AABA Form

F.7 Melodic Continuity in Time

Figure F.2 presents an analogous view for Grappelli's INT components that Figure F.1 did for his PC components. Each symbol's meaning is defined in Table F.2.

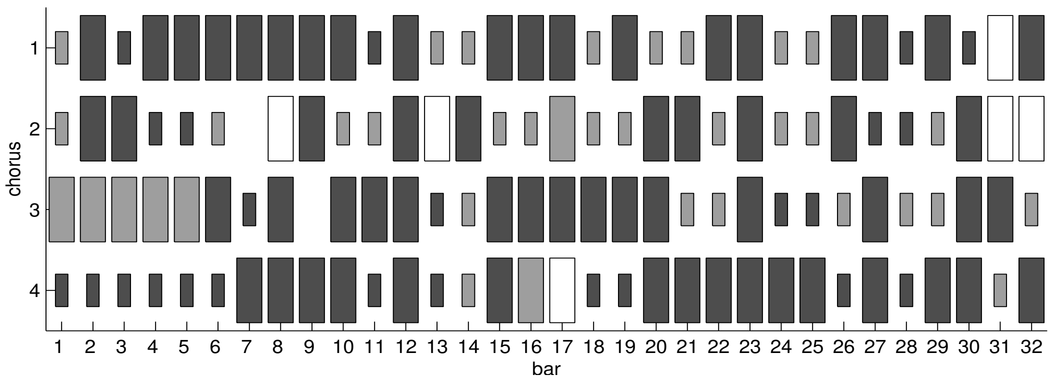


Figure F.2: Grappelli's INT Modes and the 32-bar AABA Form

F.8 Melodic Contour in Time

Figure F.3 presents an analogous view for Grappelli's DIR components that Figure F.1 did for his PC components. Each symbol's meaning is defined in Table F.3.

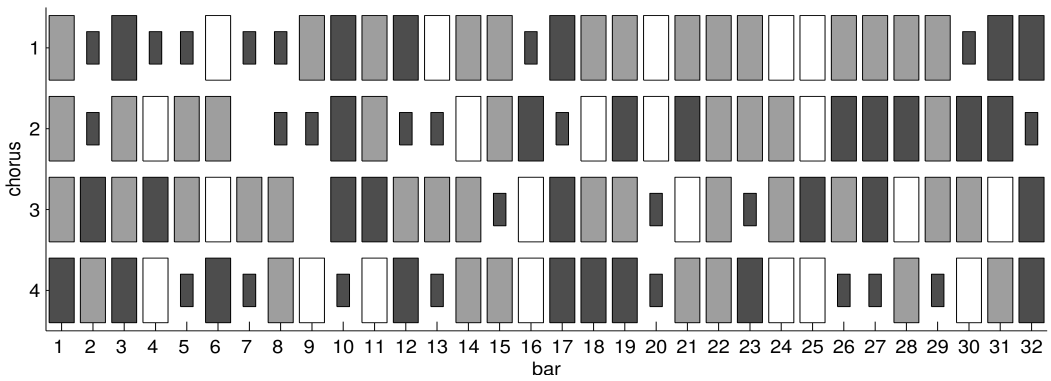


Figure F.3: Grappelli's DIR Modes and the 32-bar AABA Form

Appendix G

Listener Tests

This appendix lists the solos that were generated by BoB for the listening experiments (Section 7.7).

G.1 Parker Mode Variation Pairs

For each listening trial, a pair of call contexts was created as follows:

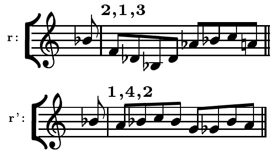
- **Select Rhythm**: randomly sample a rhythm VLT from the following set of training bars: { 7,15,27,42,48 } .
- **Select Modes**: randomly sample a pair of modes from the set:

$$\Big\{\Big\langle\langle 2,1,1\rangle,\langle 1,2,2\rangle\Big\rangle,\Big\langle\langle 4,2,2\rangle,\langle 1,4,3\rangle\Big\rangle,\Big\langle\langle 2,1,3\rangle,\langle 1,4,2\rangle\Big\rangle\Big\}.$$

- **Select Trial**: randomly sample whether or not the calls should have the same or different modes.
- **Generate**: the following generation setup was used to generate each solo: $\alpha = \alpha' = 0$; $\alpha'' = 1$; $n_{walks} = 20$; $p_{incr} = i_{incr} = 15$. In addition, Parker's absolute pitch range values were used.

For a given trial, each calls' rhythms, MRPs, and last clamps were identical. Rhythm and MRP were defined by the training bar obtained in the select rhythm step. The last clamp came from the MRP of the subsequent training bar. The mode of r was always the first element in the pair sampled in the select mode step. When the second solo was to be different, the mode of r' was the second element. The remaining subtly concerned what to do with each call's surprise. Since no call was presented to the listener, and since the goal was to determine if different and same modes could be distinguished, each mode's likelihood was set to zero (the "best" solo was the most expected one). This decision had the added benefit of reducing some of the randomness in these experiments, as did the decision to employ last clamping and hold rhythm modification constant.

Trial 1: rhythm bar 7; different

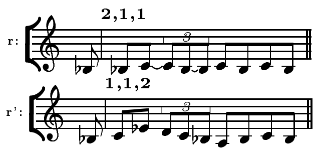


Trial 2: rhythm bar 48; different

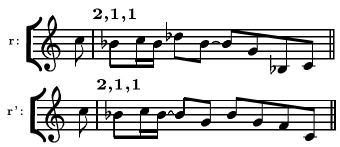
Trial 3: rhythm bar 7; same

Trial 4: rhythm bar 27; same

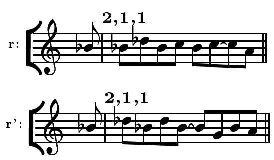
Trial 5: rhythm bar 15; different



Trial 6: rhythm bar 42; same

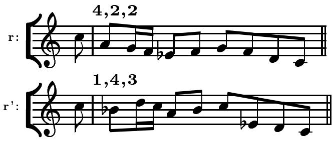


Trial 7: rhythm bar 7; same

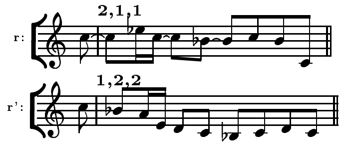


Trial 8: rhythm bar 15; different

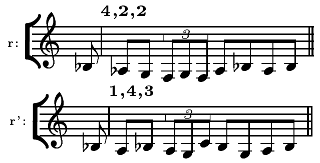
Trial 9: rhythm bar 42; different



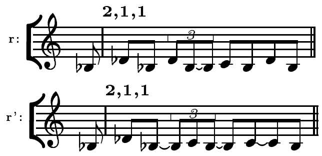
Trial 10: rhythm bar 42; different



Trial 11: rhythm bar 15; different

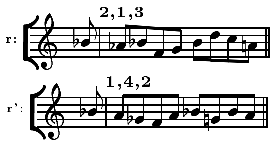


Trial 12: rhythm bar 15; same

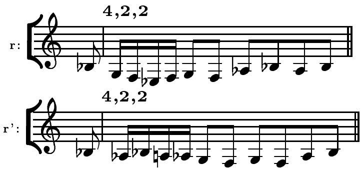


Trial 13: rhythm bar 17; same

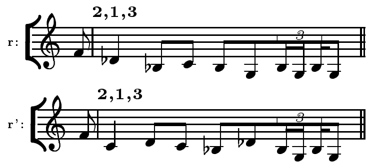
Trial 14: rhythm bar 7; different



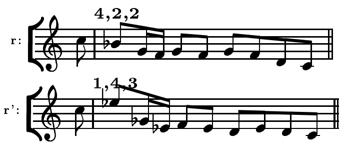
Trial 15: rhythm bar 48; same



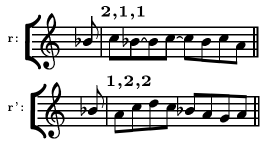
Trial 16: rhythm bar 27; same



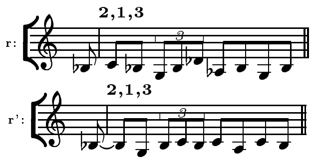
Trial 17: rhythm bar 42; different



Trial 18: rhythm bar 7; different



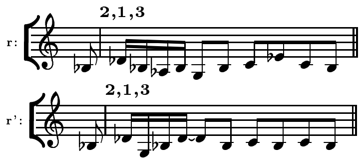
Trial 19: rhythm bar 15; same



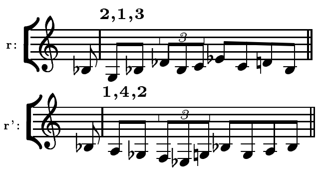
Trial 20: rhythm bar 42; same

Trial 21: rhythm bar 42; same

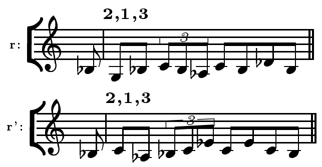
Trial 22: rhythm bar 48; same



Trial 23: rhythm bar 15; different



Trial 24: rhythm bar 15; same



Trial 25: rhythm bar 27; different

G.2 Learn Variation Pairs

For each listening trial, a pair of call contexts was created as follows:

- **Select Call**: randomly determine the start and end of the call context by sampling a tuple from a specific range of training bars. In particular, when the experiment was Grappelli based, select from the following range of bars:

$$\Big\{\langle 34, 36\rangle, \langle 92, 95\rangle, \langle 119, 122\rangle\Big\}.$$

  When the experiment was Parker based, select from the following range:

$$\left\{ \langle 25,27\rangle, \langle 42,44\rangle, \langle 74,76\rangle, \langle 106,108\rangle \right\}.$$

- **Select Trial**: randomly sample whether or not the same or different learned models should be used.
- **Generate**: the following generation setup was used to generate each solo: $\alpha = \alpha' = 0$; $\alpha'' = 1$; $n_{walks} = 20$; $p_{incr} = i_{incr} = 15$. The training data for the musician in the select call step was used to determine the absolute pitch range values.

For a given trial, each response used the same call solo, and this solo ranged over those bars in the training set defined by the select call step. Thus, for example, if the start bar was 34, the call would include what Grappelli played in bars 34 through 36, including each bars' VLTs, MRPs, and last clamps. The learned model for response r was always the musician in the select call step. What varied between trials was whether or not the same (or different) learned model was used to perceive the call's goal (surprise and mode) and generate the "in kind" response for context r'.

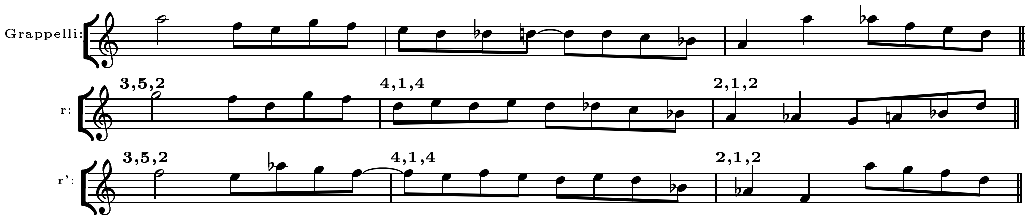
G.2.1 Grappelli Based

Trial 1: rhythm bars 34 to 36; different

Trial 2: rhythm bars 92 to 95; same

Trial 3: rhythm bars 119 to 122; same

Trial 4: rhythm bars 34 to 36; same



Trial 5: rhythm bars 92 to 95; different

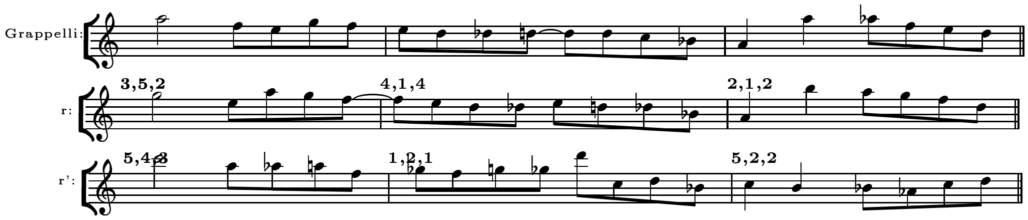
Trial 6: rhythm bars 119 to 122; different

Trial 7: rhythm bars 34 to 36; same

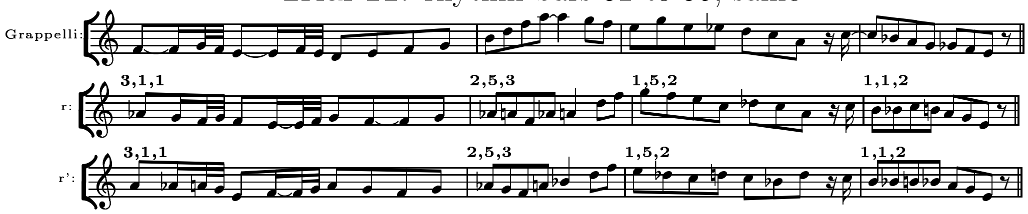
Trial 8: rhythm bars 92 to 95; same

Trial 9: rhythm bars 119 to 122; different

Trial 10: rhythm bars 34 to 36; different

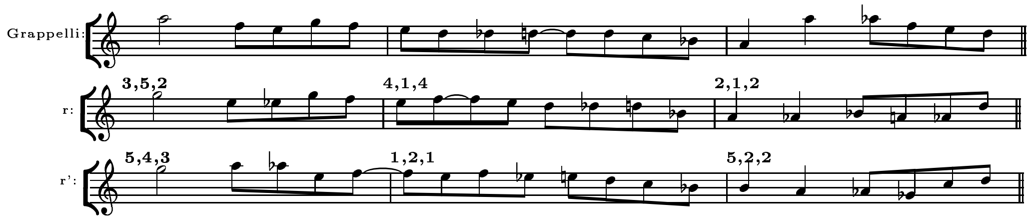


Trial 11: rhythm bars 92 to 95; same

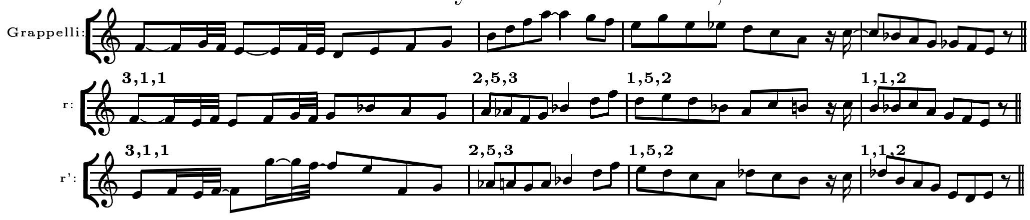


Trial 12: rhythm bars 119 to 122; same

Trial 13: rhythm bars 34 to 36; different



Trial 14: rhythm bars 92 to 95; same



Trial 15: rhythm bars 119 to 122; different

G.2.2 Parker Based

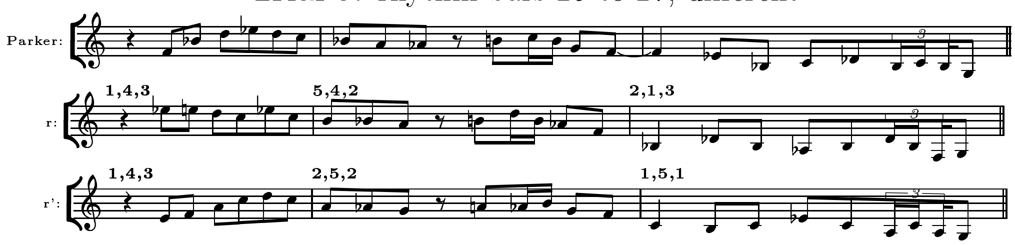
Trial 1: rhythm bars 25 to 27; different

Trial 2: rhythm bars 42 to 44; same

Trial 3: rhythm bars 106 to 108; same

Trial 4: rhythm bars 74 to 76; same

Trial 5: rhythm bars 25 to 27; different



Trial 6: rhythm bars 42 to 44; different

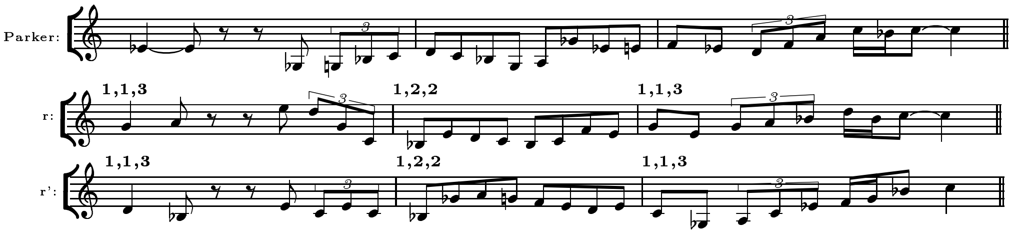
Trial 7: rhythm bars 106 to 108; same

Trial 8: rhythm bars 74 to 76; same

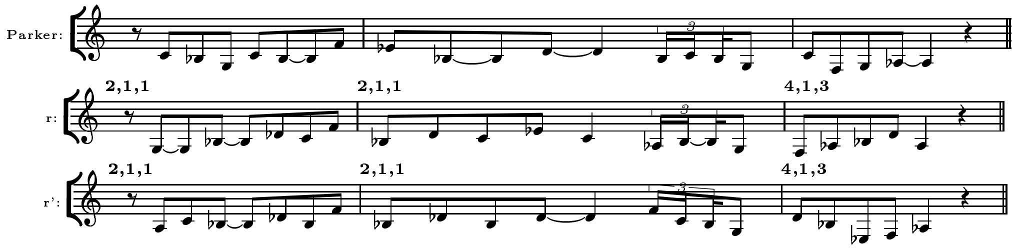
Trial 9: rhythm bars 25 to 27; different

Trial 10: rhythm bars 42 to 44; different

Trial 11: rhythm bars 106 to 108; same



Trial 12: rhythm bars 74 to 76; same



Trial 13: rhythm bars 25 to 27; different

Trial 14: rhythm bars 42 to 44; same

Trial 15: rhythm bars 106 to 108; different

Appendix H

Trading Solos

This appendix accompanies Section 7.7.5.

Four scores are transcribed below:

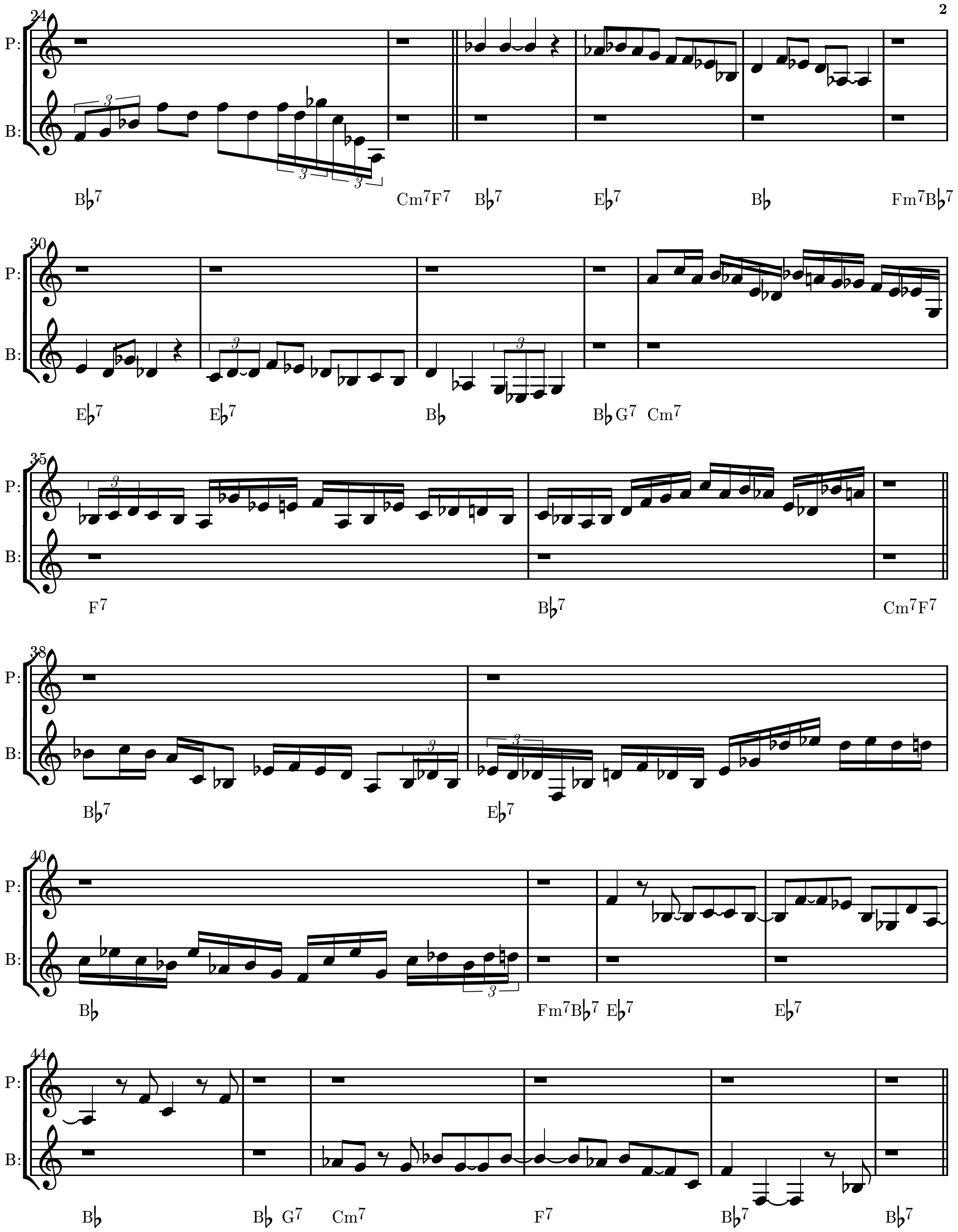
- **Figure H.1**: **Parker** trading solos with BoB trained on ***Parker***;
- **Figure H.2**: **Parker** trading solos with BoB trained on ***Grappelli***;
- **Figure H.3**: **Grappelli** trading solos with BoB trained on ***Grappelli***;
- **Figure H.4**: **Grappelli** trading solos with BoB trained on ***Parker***.

The **bold** font identifies which training set the calls are taken from and the ***bold italicized*** font identifies which training set the model has been trained on. The following setup was used to generate solos:

$$\alpha = \alpha' = 1;\ \alpha'' = 6;\ n_{walks} = 50;\ p_{incr} = i_{incr} = 12.$$

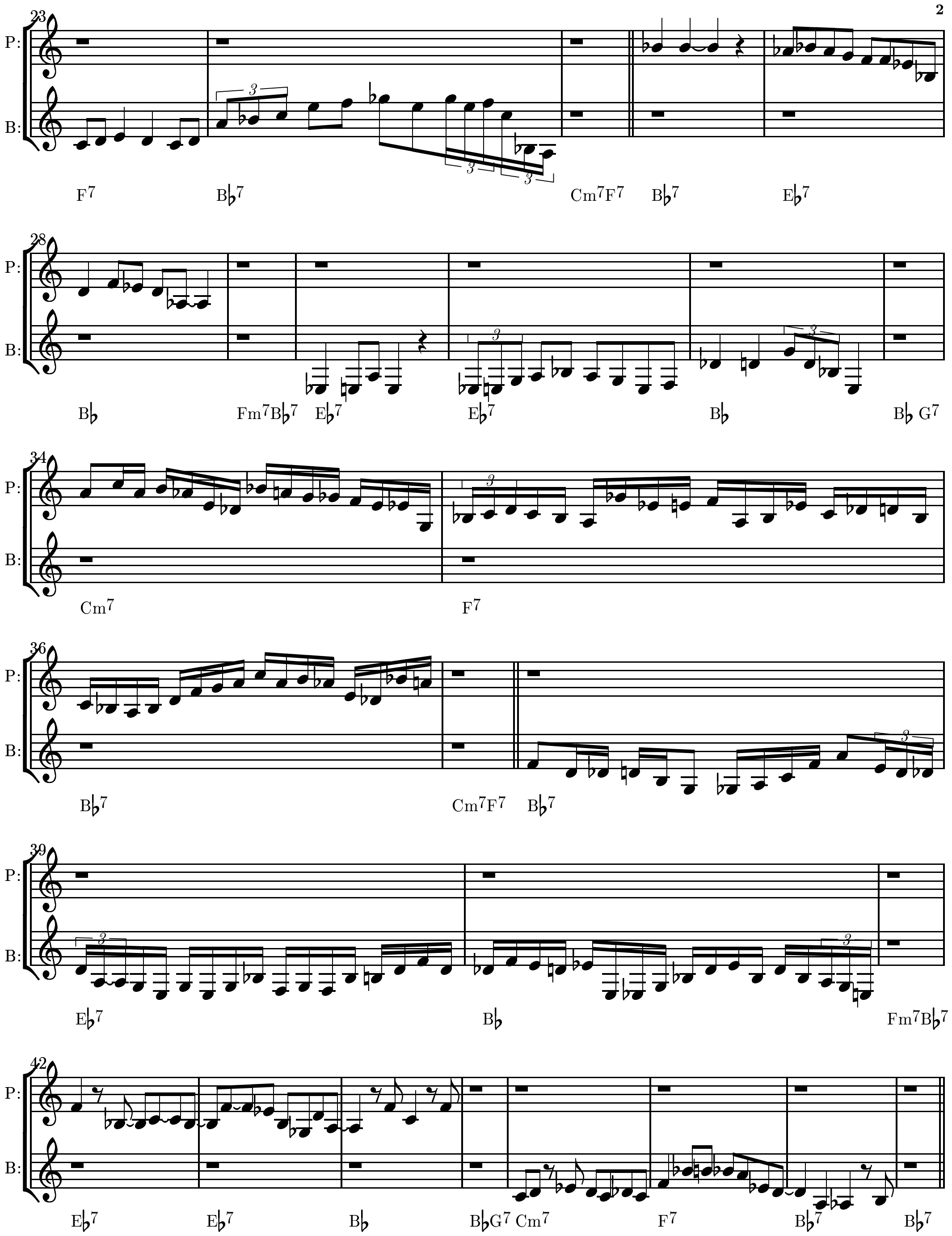
Furthermore, the call musician's absolute pitch range values were used and last clamping is not enabled. There is about a one-third chance that a particular fringe node will be modified.

Figure H.1: Parker and BoB Trained on Parker



Lily was here, 1.3.84.

Figure H.2: Parker and BoB Trained on Grappelli



Lily was here, 1.3.84.

Figure H.3: Grappelli and BoB Trained on Grappelli

1

Lily was here, 1.3.84.

Figure H.4: Grappelli and BoB Trained on Parker

Lily was here, 1.3.84.

Bibliography

Abrams, S., D. V. Oppenheim, D. Pazel, and J. Wright (1999). Higher-level composition control in MusicSketcher: Modifiers & smart harmony. In *Proceedings of the 1999 ICMC*. International Computer Music Association.

Aebersold, J. (1992). *How to Play Jazz and Improvise*. Jamey Aebersold, New Albany, IN.

Arcos, J., R. L. de Mantaras, and X. Serra (1998). Saxex: A case-based reasoning system for generating expressive musical performances. *Journal of New Music Research 27*(3), 194–210.

Assayag, G., C. Rueda, M. Laurson, C. Agon, and O. Delerue (1998). Guessing the composer's mind: Applying universal prediction to musical style. In *Proceedings of the 1998 ICMC*. International Computer Music Association.

Association, I. M. (1989). MIDI 1.0 specification.

AudioWorks (1988-1999). Sound2Midi. `http://www.audioworks.com/s2m/s2m.htm`.

Bailey, D. (1992). *Improvisation, Its Nature & Practice in Music*. Da Capo Press.

Baker, D. (1983). *Jazz improvisation : A Comprehensive Method of Study For All Players*. Frangipani Press.

Bartlett, J. C. (1993). Tonal structure of melodies. In T. J. Tighe and W. J. Dowling (Eds.), *Psychology & Music: the Understanding of Melody & Rhythm*. Lawrence Erlbaum Associates, Inc.

Bates, J. (1992). The nature of character in interactive worlds. Technical Report CMU-CS-92-200, School of Computer Science, Carnegie Mellon University.

Bellgard, M. (1993). *Machine Learning of Constraints in Temporal Sequences: Applications of the Effective Boltzmann Machine*. Ph. D. thesis, Computer Science Department, University of Western Australia.

Berliner, P. F. (1994). *Thinking in Jazz, The Infinite Art of Improvisation.* University of Chicago Press.

Bharucha, J. J. (1994). Tonality and expectation. In R. Aiello and J. A. Sloboda (Eds.), *Musical Perceptions.* Oxford University Press.

Biles, J. (1994). GenJam: A genetic algorithm for generating jazz solos. In *Proceedings of the 1994 ICMC.* International Computer Music Association.

Biles, J. (1995). Training an iga via audience-mediated performance. In *Proceedings of the 1995 ICMC.* International Computer Music Association.

Biles, J. (1996). Neural network fitness functions for an iga. In *Proceedings of the International ICSC Symposium on Intelligent Industrial Automation and Soft Computing.* ICSC Academic Press.

Biles, J. (1998). Interactive GenJam: Integrating real-time performance with a genetic algorithm. In *Proceedings of the 1998 ICMC.* International Computer Music Association.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition.* Clarendon Press.

Boltz, M. G. (1999). The processing of melodic and temporal information: Independent or unified dimensions? *Journal of New Music Research 28*(1), 66–79.

Bresin, R. and A. Friberg (2000). Emotional coloring of computer-controlled music performances. *Computer Music Journal 24*(4), 44–63.

Brooks, R. A. (1990). Elephants don't play chess. *Robotics and Autonomous Systems 6.*

Bryson, J. J. (1992). The subsumption strategy development of a music modeling system. Master's thesis, Dept of Artificial Intelligence, University of Edinburgh.

Cadez, I., S. Gaffney, and P. Smyth (2000). A generalized probabilistic framework for clustering individuals. Technical Report UCI-ICS-00-09, University of Irvine, Information and Computer Science.

Cadez, I. V. and P. Smyth (2001). Model complexity, goodness of fit and diminishing returns. In *Advances in Neural Information Processing Systems 13 (to appear).* MIT Press.

Cambouropoulos, E. (2001). The local boundary detection model (LBDM) and its application in the study of expressive timing. In *Proceedings of the ICMC 2001.* ICMC.

Cambouropoulos, E., T. Crawford, and C. S. Iliopoulos (1999). Pattern processing in melodic sequences: Challenges, caveats & prospects. In *Proceedings from the AISB'99 Symposium on Musical Creativity.*

Casella, G. and R. L. Berger (1990). *Statistical Inference.* Wadsworth, Belmont, CA.

Casserly, L. (1997). Person to ... person? *Resonance Magazine 6*(1).

Cemgil, A. and B. Kappen (2001a). Bayesian real-time adaptation for interactive performance systems. In *Proceedings of the 2001 ICMC.* International Computer Music Association.

Cemgil, A. and B. Kappen (2001b). Tempo tracking and rhythm quantization by sequential monte carlo. In *Advances in Neural Information Processing Systems 13 (to appear).* MIT Press.

Cemgil, A. T., P. Desain, and B. Kappen (2000). Rhythm quantization for transcription. *Computer Music Journal 24*(2), 60–76.

Chvatal, V. (1983). *Linear Programming.* W.H. Freeman.

Coker, J. (1970). *The Jazz Idiom.* Prentice-Hall Inc., Englewood Cliffs, NJ.

Cope, D. (1992). Computer modeling of musical intelligence in EMI. *Computer Music Journal 16*(2), 69–83.

Cross, I. (1999). Ai and music perception. *AISB Quarterly 102*, 12–25.

Cunha, U. S. and G. Romalho (2000). Combining off-line and on-line learning for predicting chords in tonal music. In *Proceedings from the AAAI-2000 Music and AI Workshop.* AAAI Press.

Dannenberg, R. B. (1993). Software design for interactive multimedia performance. *Interface - Journal of New Music Research 22*(3), 213–218.

Dannenberg, R. B. and I. Derenyi (1998). Combining instrument and performance models for high-quality music synthesis. *Journal of New Music Research 27*(3).

Dannenberg, R. B. and B. Mont-Reynaud (1987). Following an improvisation in real time. In *Proceedings of the 1987 ICMC.* International Computer Music Association.

Dannenberg, R. B., B. Thom, and D. Watson (1997). A machine learning approach to musical style recognition. In *Proceedings of the 1997 ICMC.* International Computer Music Association.

DeGroot, M. H. (1970). *Optimal statistical decisions.* McGraw-Hill.

Dempster, A. P., N. M. Laird, and D. B. Rubin (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*.

Desain, Aarts, Cemgil, Kappen, and van Thienen (1999). Robust time-quantization for music, from performance to score. In *Proceedings of the 106th AES Convention.* Audio Engineering Society.

Deutsch, D. (1999). The psychology of music. Academic Press.

Dubes, R. and A. Jain (1980). Clustering methodologies in exploratory data analysis. *Advances in Computers 19*(11).

Dubnov, S., G. Assayag, and R. El-Yaniv (1998). Universal classification applied to musical sequences. In *Proceedings of the 1998 ICMC.* International Computer Music Association.

Duda, R. O. and P. E. Hart (1973). *Pattern classification and scene analysis.* Wiley.

Elliot, C. and J. Brzezinski (1998). Autonomous agents as synthetic characters. *AI Magazine*, 13–30.

Elman, J. (1991). Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning 7*(2-3), 195–225.

Feulner, J. and D. Hörnel (1994). Melonet: Neural networks that learn harmony-based melodic variations. In *Proceedings of the 1994 ICMC.* International Computer Music Association.

Frank, A., A. Stern, and B. Resner (1997). Socially intelligent Virtual Petz. In K. Dautenhahn (Ed.), *Proceedings of the 1997 AAAI Fall Symposium on Socially Intelligent Agents.* AAAI Press.

Franklin, J. A. (2001). Multi-phase learning for jazz improvisation and interaction. In *Proceedings of the Eighth Biennial Symposium for Arts & Technology.*

Giomi, F. and M. Ligabue (1988). An interactive system for musical improvisation. In *Proceedings of the 1988 ICMC.* International Computer Music Association.

Glaser, M. and S. Grappelli (1981). *Jazz Violin.* Oak Publications, NY, NY.

Goldsen, M. H. (Ed.) (1978). *Charlie Parker Omnibook: For C Instruments.* Atlantic Music Corp.

Gridley, M. C. (1988). *Jazz Styles, History and Analysis* (third ed.). Prentice Hall.

Hanson, R., J. Stutz, and P. Cheeseman (1991). Bayesian classification theory. Technical Report FIA-90-12-7-01, NASA Ames Research Center.

Hayes-Roth, B. and R. van Gent (1997). Story-making with improvisational puppets. In W. Johnson (Ed.), *Proceedings of the First International Conference on Autonomous Agents*, pp. 1–7.

Heijink, H., P. Desain, H. Honing, and W. Windsor (2000). Make me a match: An evaluation of different approaches to score-performance matching. *Computer Music Journal*, 43–56.

Hild, H., J. Feulner, and W. Menzel (1991). Harmonet: A neural net for harmonizing chorales in the style of J.S. Bach. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky (Eds.), *Advances in Neural Information Processing 4*, pp. 267–274. Morgan Kaufman.

Hörnel, D. (1998). A multi-scale neural-network mode for learning and reproducing chorale variations. In W. B. Hewlett and E. Selfridge-Field (Eds.), *Melodic Similarity; Concepts, Procedures, and Applications*. The MIT Press.

Hörnel, D., J. Langnickel, B. Sandberger, and B. Sieling (1999). Statistical vs. connectionist models of Bebop improvisation. In *Proceedings of the 1999 ICMC*. International Computer Music Association.

Hörnel, D. and W. Menzel (1998). Learning musical structure & style with neural networks. *Computer Music Journal 22*(4), 44–62.

Höthker, K. (1999). Modeling the motivic process of melodies with markov chains. In *Proceedings of the 1999 ICMC*, pp. 383–386.

Johnson-Laird, P. N. (1991). Jazz improvisation: A theory at the computational level. In P. Howell (Ed.), *Representing Musical Structure*, pp. 291–325. Academic Press, Inc.

Kaufman, L. and P. J. Rousseeuw (1990). *Finding groups in data : an introduction to cluster analysis*. Wiley series in probability and mathematical statistics. Wiley.

Kearns, M., Y. Mansour, and A. Ng (1997). An information-theoretic analysis of hard and soft assignment methods for clustering. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, pp. 282–293. Morgan Kaufmann.

Kozen, D. C. (1991). *The design and analysis of algorithms*. Springer-Verlag.

Krumhansl, C. M. (1990). *Cognitive Foundations of Musical Pitch*. Oxford University Press.

Lartillot, O., S. Dubnov, G. Assayag, and G. Bejerano (2001). Automatic modeling of musical style. In *Proceedings of the 2001 ICMC*. International Computer Music Association.

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

Levitt, D. (1981). A melody description system for jazz improvisation. Master's thesis, MIT.

Longuet-Higgens, H. (1987). *Mental Processes*. MIT Press.

Loy, G. (1991). Connectionism and musiconomy. In P. M. Todd and D. G. Loy (Eds.), *Music and Connectionism*, pp. 23–36. MIT Press.

Loyall, B. (1997). *Believable Agents: Building Interactive Personalities*. Ph. D. thesis, Computer Science Department, CMU.

Marsden, A. (1999). Representation of melodies as elaboration trees. In *Proceedings from the AISB'99 Symposium on Musical Creativity*.

Mateas, M. (1999). An oz-centric review of interactive drama & believable agents. In M. J. Wooldridge and M. M. Veloso (Eds.), *Lecture Notes in Artificial Intelligence*, Volume 1600, pp. 297–329. Springer-Verlag.

Mateas, M. (2001a). Expressive AI. *Leonardo: The Journal of the International Society of the Arts, Sciences and Technology 34*(2), 147–153.

Mateas, M. (2001b). Interactive drama, art and artificial intelligence. Unpublished Dissertation Proposal.

McCorduck, P. (1991). *Aaron's Code: Meta-art, Artificial Intelligence, and the Work of Harold Cohen*. W.H. Freeman.

McLachlan, G. J. and K. E. Basford (1988). *Mixture Models: Inference & Applications to Clustering*. Marcel Dekker.

Meila, M. and D. Heckerman (1998). An experimental comparison of several clustering and initialization methods. Technical Report MSR-TR-98-06, Microsoft Research Center.

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

Minsky, M. (1986). *The Society of Mind*. Simon and Schuster.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.

Mongeau, M. and D. Sankoff (1990). Comparison of musical sequences. *Computers and the Humanities 24*, 161–175.

Mozer, M. C. (1994). Neural network music composition by prediction: Exploring the benefits of psychophysical constraints and multi-scale processing. *Connection Science 6*, 247–280.

Narmour, E. (1990). *The Analysis and Cognition of Basic Melodic Structures*. University of Chicago Press.

Nienhuys, H.-W., J. Nieuwenhuizen, and A. Mariano (1999). GNU lilypond, the music typesetter. `http://www.lilypond.org`.

Nigam, K., A. McCallum, S. Thrun, and T. Mitchell (1998). Learning to classify text from labeled & unlabeled documents. In *Proceedings of the 1998 AAAI*. AAAI Press.

Ostransky, L. (1977). *Understanding Jazz*. Prentice-Hall, Inc., Englewood Cliffs, NJ.

Pachet, F. (2000). Computer analysis of jazz chord sequence: Is *Solar* a blues? In E. R. Miranda (Ed.), *Readings in Music and Artificial Intelligence*. Harwood Academic Publishers.

Papadopoulos, G. and G. Wiggins (1999). Ai methods for algorithmic composition: A survey, a critical view and future prospects. In *Proceedings from the AISB'99 Symposium on Musical Creativity*.

Pennycook, B. and D. Stammen (1993). Real-time recognition of melodic fragments using the dynamic timewarp algorithm. In *Proceedings of the 1993 ICMC*. International Computer Music Association.

PGMusic (1999). Band-in-a-box. `http://www.pgmusic.com`.

Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery (1992). *Numerical recipes in C : the art of scientific computing*. Cambridge University Press.

Puckette, M. and D. Zicarelli (1991). MAX development package manual. Opcode Systems, Inc.

Rabiner, L. and B. Juang (1986). An introduction to hidden markov models speech recognition and processing. *IEEE ASSP Magazine 3*(1), 4–16.

Ramoni, M., P. Sebastiani, and P. R. Cohen (2000a). Multivariate clustering by dynamics. In *Proceedings of the AAAI/IAAI 2000*, pp. 633–638.

Ramoni, M., P. Sebastiani, and P. R. Cohen (2000b). Unsupervised clustering of robot activities: a bayesian approach. In *Proceedings of the Fourth International Conference on Autonomous Agents*, pp. 134–135.

Raphael, C. (1999). Automatic segmentation of acoustic musical signals using hidden markov models. *IEEE Transactions on Pattern Analysis & Machine Intelligence 10*(4).

Raphael, C. (2001). Music plus one: A system for flexible and expressive musical accompaniment. In *Proceedings of the 2001 ICMC*. International Computer Music Association.

Reis, B. Y. (1999). Simulating music learning: On-line perceptually guided pattern induction of context models for multiple-horizon prediction of melodies. In *Proceedings from the AISB'99 Symposium on Musical Creativity*.

Roads, C. (1985). Improvisation with George Lewis. In C. Roads (Ed.), *Composers and the Computer*, pp. 75–88. William Kaufmann, Inc.

Rolland, P. and J. Ganascia (1996). Automated motive-oriented analysis of musical corpuses: a jazz case study. In *Proceedings of the 1996 ICMC*. International Computer Music Association.

Rolland, P.-Y. and J.-G. Ganascia (2000). Musical pattern extraction and similarity assessment. In E. R. Miranda (Ed.), *Readings in Music and Artificial Intelligence*, pp. 115–144. Harwood Academic Publishers.

Ross, S. M. (1993). *Introduction to Probability Models*. Academic Press Ltd.

Rowe, R. (1993). *Interactive Music Systems : Machine Listening & Composing*. MIT Press.

Rowe, R. (2000). Interactive music systems in ensemble performance. In E. R. Miranda (Ed.), *Readings in Music and Artificial Intelligence*. Harwood Academic Publishers.

Russell, G. (1959). *The Lydian Chromatic Concept of Tonal Organization for Improvisation: For All Instruments*. Concept Publishing, Cambridge, MA.

Sebastiani, P., M. Ramoni, and P. Cohen (2000). Sequence learned via bayesian clustering by dynamics. Technical Report KMi-TR-89, Knowledge Media Institute.

Sengers, P. (1998). *Anti-Boxology: Agent Design in a Cultural Context*. Ph. D. thesis, Computer Science Department, CMU.

Smyth, P. (1997). Clustering sequences with hidden markov models. In *Advances in Neural Information Processing Systems 9*. MIT Press.

Smyth, P. (1999). Probabilistic, model-based clustering of multivariate and sequential data. In *Proceedings of the 7th International Workshop on AI and Statistics*. Morgan Kaufman.

Spiegel, M. R. (1994). *Theory & Problems of Probability & Statistics*. Schaum's Outline. McGraw-Hill.

Stammen, D. and B. Pennycook (1994). Real-time segmentation of music using an adaptation of lerdahl and jackendoff's grouping principles. In *Proceedings of the 1994 ICMPC*.

Sudnow, D. (1978). *Ways of the Hand, the Organization of Improvised Conduct*. Harvard University Press.

Thom, B. (1995). Predicting chords in jazz: the good, the bad & the ugly. In *Working Notes of the IJCAI Workshop on AI and Music*, Montreal. International Joint Conferences on Artificial Intelligence.

Thom, B. (1999). Learning melodic models for interactive melodic improvisation. In *Proceedings of the 1999 ICMC*. International Computer Music Association.

Thom, B. (2000a). Artificial intelligence and real-time interactive improvisation. In *Proceedings from the AAAI-2000 Music and AI Workshop*. AAAI Press.

Thom, B. (2000b). Unsupervised learning and interactive jazz/blues improvisation. In *Proceedings of the AAAI-2000*. AAAI Press.

Toiviainen, P. (1995). Modeling the target-note technique of bebop-style improvisation. *Music Perception 12*(4), 399–413.

Trilsbeek, P. and H. v. Thienen (1999). Quantization for notation: methods used in commercial music software. Technical report, University of Nijmegen.

Vapnick, V. (1982). *Estimation of Dependences Based on Empirical Data*. Springer-Verlag.

Walker, W. (1997). A computer participant in musical improvisation. In *CHI 97 Electronic Publications.*

Weigend, A. and N. Gershenfeld (1993). *Results of the time series prediction competition at the Santa Fe Institute.* IEEE.

Wessel, D. and M. Wright (2000). Problems and prospects for intimate musical control of computers. In *CHI '01 Workshop New Interfaces for Musical Expression.* ACM SIGCHI.

Wessel, D., M. Wright, and S. A. Kahn (1998). Preparation for improvised performance in collaboration with a Khyal singer. In *Proceedings of the 1998 ICMC.* International Computer Music Association.

Weyrauch, P. (1997). *Guiding Interactive Drama.* Ph. D. thesis, Computer Science Department, CMU.

Widmer, G. (1996). Recognition & exploitation of contextual clues via incremental meta-learning. In *Proceedings of the 1996 ICML.*

Widmer, G. (2000). Large-scale induction of expressive performance rules: First quantitative results. In *Proceedings of the 2000 ICMC.* International Computer Music Association.

Winkler, T. (1998). *Composing Interactive Music: Techniques and Ideas Using MAX.* MIT Press.

Yoo, L. and I. Fujinaga (1999). A comparative latency study of hardware and software pitch-trackers. In *Proceedings of the 1999 ICMC.* International Computer Music Association.