UNIVERSITY OF CALIFORNIA
SANTA CRUZ

MODELING EXPRESSIVE MUSICAL PERFORMANCE WITH HIDDEN MARKOV MODELS

A dissertation submitted in partial satisfaction of the
requirements for the degree of

Master of Science

in

COMPUTER SCIENCE

by

Graham Charles Grindlay

March 2005

The Dissertation of Graham Charles Grindlay
is approved:

Contents

List of Figures **iv**

Abstract **v**

1 Introduction **1**

2 Related Work **4**

3 The ESP System **7**

3.1 Musical Features . . . 8

3.1.1 Melodic Features . . . 9

3.1.2 Accompaniment Features . . . 11

3.2 Representing the Data . . . 14

3.2.1 Discrete-valued Features . . . 15

3.2.2 Continuous-valued Features and the Kernel Technique . . . 16

3.3 Training . . . 18

3.3.1 The Entropic MAP Estimator . . . 18

3.3.2 Trimming . . . 20

3.3.3 Annealing . . . 22

3.4 Synthesis . . . 24

4 Experiments **29**

5 Conclusion **31**

A Hidden Markov Models **34**

A.1 Introduction . . . 34

A.2 Observations . . . 36

A.3 Training . . . 37

B Deriving the Entropic MAP Estimator **41**

Bibliography **45**

List of Figures

1.1 A "pianoroll" representation of how a performer might interpret and deviate from a written score (note that this representation does not contain velocity information). The vertical position of each bar indicates the pitch of each note, the horizontal position indicates when the note was played, and the length of the bar indicates for how long it was played. *Top*: a human performance of a simple melody line. *Center*: a literal rendition of the same melody according to the score. *Bottom*: an overlapped representation, showing when and how the performed rendition differs from that of the score. 2

3.1 An example of a distribution represented by a set of 300 kernels. Although a mixture-of-Gaussians could be used to fit this distribution to varying degrees of accuracy, much of the fine detail would be lost. 17

3.2 Here we show the energy surface of a probability distribution and how it varies as we anneal it. Green represents the value of the uniform distribution (which in this plot is .01), red indicates a value higher than that of the uniform, and blue indicates a lower value. 23

A.1 A simple 4-state HMM that outputs three types of symbols: "a", "b", and "c". Clear circular nodes represent hidden states while shaded square nodes represent observables. Transitions and observations are labeled with their probabilities and the arrow at the top gives the prior probability of starting in each state. 35

B.1 This plot shows the real-valued branches of the Lambert W function. The W function is the multivalued inverse of $w \rightarrow we^w$ and so we can plot the real branches indirectly, by plotting $w \rightarrow we^w$ with the axes swapped. 44

Abstract

Modeling Expressive Musical Performance with Hidden Markov Models

by

Graham Charles Grindlay

Although one can easily produce a literal audio rendition of a musical score, the result is often bland and emotionally dry. In this thesis, we consider the problem of modeling and synthesizing expressive piano performance. Expressive piano performance can be characterized by variations in three primary dimensions: note duration, note articulation (the gap or overlap between consecutive notes), and note velocity (loudness). It is largely the manner in which a performer chooses these score variations that determines the character of a performance.

Hidden Markov models provide a flexible and efficient statistical framework for this problem domain. We describe a novel system, based on hidden Markov models, which is capable of learning the ways in which compositional structure (i.e. a score) relates to music performance in a corpus of training data. Using a set of features to represent notes in both scores and performances, our system automatically identifies statistically salient musical contexts and associates each of these with a state in the HMM. Each state models the distribution of score and performance features appropriate for that context. Once trained, the system can synthesize renditions of novel scores in the style of the performances used in training.

We also report on a series of informal listening tests which indicate that our system is capable of producing score renditions that are not only preferable to literal audio renditions, but can even approach the quality of human performance in some cases.

Chapter 1

Introduction

A large part of what makes music such an enjoyable and emotionally powerful phenomenon comes from the manner in which it is performed. Subtle (and sometimes not-so-subtle) deviations from the score in timing, duration, and articulation (see Figure 1.1), help to color the emotional character of a piece. There is a great deal of agreement between performers in terms of what types of score deviations are appropriate for a given musical context. Factors such as motor constraints, similarities in cognitive processing, and cultural norms, all contribute in defining the common model of musical performance. However, while there may be general agreement on performance practices, individual performers often exhibit a great degree of variation in their execution of these performance strategies. In fact, oftentimes there is even a fair amount of variability amongst performances by a single performer. Nonetheless, it is our belief (and our experiments indicate this to be so) that despite these intra-performer variations, there are enough commonalities to make it possible to derive a general model of musical performance.

In this thesis, we describe the Expressive Synthetic Performance (ESP) system, a system capable of modeling and synthesizing stylistic piano performance in an efficient and fully automated fashion. The ESP system is based on a type of unsupervised machine learning

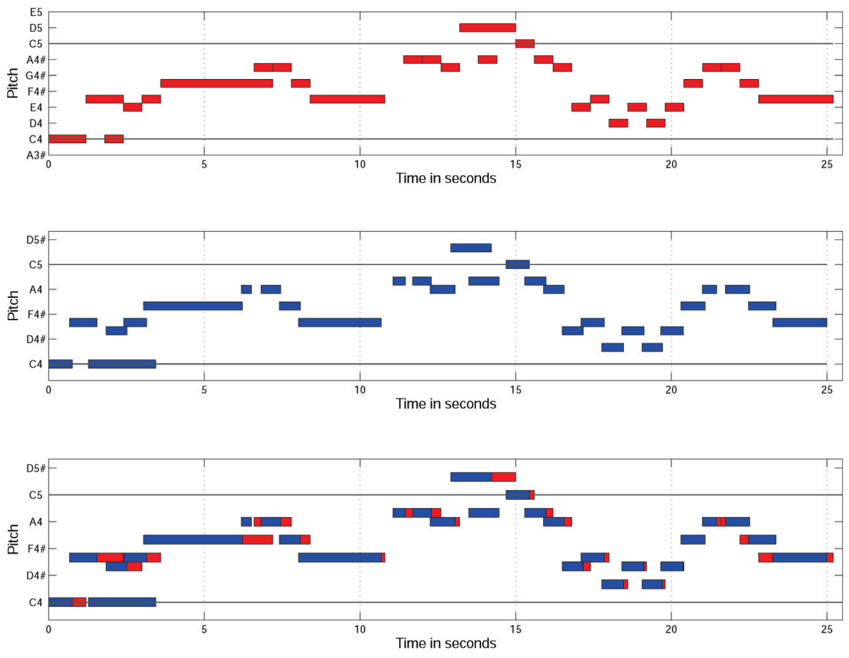

Figure 1.1: A "pianoroll" representation of how a performer might interpret and deviate from a written score (note that this representation does not contain velocity information). The vertical position of each bar indicates the pitch of each note, the horizontal position indicates when the note was played, and the length of the bar indicates for how long it was played. *Top*: a human performance of a simple melody line. *Center*: a literal rendition of the same melody according to the score. *Bottom*: an overlapped representation, showing when and how the performed rendition differs from that of the score.

technique called a hidden Markov model (HMM). HMMs provide a convenient framework for our problem domain as they are flexible, have efficient training algorithms, and benefit from a rich and well-studied theoretical background.

Our system identifies statistically salient musical contexts present in the training data as well as the types of score deviations appropriate for those contexts. Musical contexts are represented by states in the HMM, while appropriate score deviations are represented probabilistically by both multinomial and Gaussian distributions local to each state. These distributions describe, for each state, the joint probability of observing a particular type of scored note and a particular way of performing that note. Training proceeds via a modified version of the Expectation-Maximization [15] algorithm due to Brand [4] which, instead of solely trying to increase the likelihood of the model parameters at each iteration, seeks a maximum *a posteriori* solution that increases the model likelihood while balancing the desire that the model parameters be sparse (i.e. have low entropy). Once a model has been trained, it can be used to synthesize stylized renditions of novel scores. This is done by first finding a Viterbi [34] path through the model (a sequence of states that maximizes the marginal probability of the novel score's features with respect to the model) and then using the Expectation-Maximization algorithm to find a sequence of performance features that maximizes the likelihood of the model given the new score.

Currently, the system has been implemented for both piano melodies as well as melodies with chordal accompaniment. However, the addition of accompaniment support is a relatively recent effort, bringing with it a host of questions regarding the best representational scheme. We will discuss our approach to both types of modeling and synthesis, but given the somewhat preliminary state of polyphonic support, we will concentrate the discussion on our approach to melodic modeling. However, it is important to note that the ESP system framework is indifferent to the data being modeled and therefore success in modeling fully polyphonic music should depend almost entirely on finding an appropriate feature set, rather than on requiring any substantial changes to the system itself.

We begin by reviewing the literature relevant to our problem domain. We then discuss the ESP system in some detail, first covering feature sets and then the specifics of the training and synthesis algorithms. Next we discuss some experimental results obtained from informal listening tests. Finally, we summarize our findings and conclude with some ideas for the future. For readers not already familiar with hidden Markov models, we provide a brief tutorial in Appendix A.

Chapter 2

Related Work

Artificial intelligence and machine learning techniques have found a variety of applications to musical problems in recent years. Much of this work has centered around the so-called "transcription problem" (trying to extract the symbolic representation of a piece of music from raw audio) [28, 12, 30, 16] and the related sub-problems of tempo tracking and rhythm quantization [11]. However, the problems of modeling aspects of compositional and performance style has received somewhat less attention.

Cope models compositional style with his *Experiments in Musical Intelligence* software [13, 14]. This expert system extracts what he refers to as "signatures"; common compositional units employed by a composer across multiple works. These "signatures" are then used, along with external constraints on compositional structure, to produce novel scores in the style of a particular composer.

Widmer has worked to model and synthesize expressive performances of piano melodies using a system that automatically learns rules from musical data [37]. This system extracts sets of simple rules, each of which partially explains an aspect of the relationship between score and performance. These simple rules are then combined into larger rules via clustering, generalization, and heuristic techniques.

Widmer and Tobudic describe a multi-scale representation of piano tempo and dynamics curves which they use to learn expressive performance strategies [38]. This approach abstracts score deviation temporally and learns how these deviations are applied at both the note and phrase levels via rule-induction algorithms.

Archos and López de Mántaras use non-parametric case-based reasoning in their *SaxEx* system [1] which renders expressive saxophone melodies from examples. Their system utilizes a database of transformations from scores to expressive performances with features such as dynamics, rubato, vibrato, and articulation. *SaxEx* synthesizes new performances by looking in its database for similar score features and then applying the corresponding expressive transformations. Conflicts that arise when matching score features to entries in the database are resolved with heuristic techniques.

Bresin, Friberg, and Sundberg use a rule-based approach in *Director Musices*, a system for creating expressive performances from raw scores [8]. This system is based on a set of handcrafted rules for synthesizing piano performances. These rules are parametric, providing a means for tuning the contribution of each rule in the synthesis of an expressive performance.

Although most research on applications of machine learning to expressive performance has focused almost exclusively on generative models, Stamatatos and Widmer describe a system for classifying performances [31]. This system maintains a database of performers and associates each with certain types of performance features. Then, given a performance, the system uses an ensemble of simple classification algorithms to identify the most likely performer.

Hidden Markov models (HMMs) are a powerful subclass of graphical models and are used for learning and synthesizing sequence data in a variety of applications. We provide a brief tutorial in Appendix A for those readers unfamiliar with HMMs. For a more extensive treatment, the reader is referred to Rabiner's excellent tutorial [27].

Raphael uses Bayesian networks (a generalization of HMMs) in his *Music Plus One* system [29]. The *Play* component of *Music Plus One* uses a Bayesian network to learn performer-specific expressive accompaniment. The structure of the Bayesian network is constructed using the score and then trained with a series of "rehersals". After training, *Play* can provide expressive real-time accompaniment for the piece on which it was trained.

Wang *et al.* [35] use HMMs to learn and synthesize a conductor's movements. Their work uses a single HMM where each state models a joint distribution over the music (energy, beat, and pitch features) and the positions and velocities of markers on the joints of a conductor. Wang *et al.* build on earlier work by Brand using HMMs to synthesize lip movements matching speech [5] and by Brand and Hertzmann to model different styles of bipedal movement [6]. Whereas the above approaches use one HMM with dual outputs, Casey [10] converts between different drum styles by using a pair of HMMs and a mapping between their states.

Chapter 3

The ESP System

The goal of the ESP system is to provide a flexible and efficient framework with which to model and synthesize expressive renditions of musical scores. In contrast to other approaches to this type of problem, we use statistical learning techniques. There are several attractive properties of this family of algorithms, including the ability to efficiently handle large amounts of complex data, resistance to noise, and flexibility. Unlike a purely rule-based approach, our system is fully unsupervised and therefore needs no prior knowledge.

The basic setup of our problem is as follows: we are given a set of training performances and their corresponding scores. We wish to train a hidden Markov model on these example score-performance pairs, capturing the essential structure that describes their relationships. To do so, we first extract from each example in the training set, a sequence S of score feature vectors and the corresponding sequence R of rendition feature vectors. Each feature vector, $s \in S$, corresponds to one note in the score data while each feature vector $r \in R$, corresponds to one note in the performance data. Once the model has been trained we can use the synthesis algorithm to render expressive performances of new scores.

3.1 Musical Features

As with all learning techniques, both the data and the way in which it is represented play a fundamental role in the performance of the ESP system. Although theories of musical cognition [23, 24, 25] may be able to provide better representations, one of our goals for the ESP system is to keep it as automated as possible. Therefore we currently restrict ourselves to a relatively simple feature set. However, given the importance of data representation, we believe that this area of the system warrants future attention.

The ESP system requires a symbolic representation of music for training. Currently, we use MIDI[1], a widely used format standard for encoding the information necessary to describe what notes to play, when to play them, and to some extent, how to play them. [2] Our training data was extracted directly from a Disklavier piano, thus avoiding the difficult transcription problem. [3] Because our performance features make use of comparisons between how a note was scored and how it was actually played, we must contend with the problem of missing, additional, and incorrectly played notes in the performance data. We do this automatically by finding the best matching between score and performance notes using an algorithm proposed by Bloch and Dannenberg [3]. This algorithm, which is based on dynamic programming, is somewhat reminiscent of the standard longest-common subsequence algorithm. From the matched data, we extract two sets of features, one for the score data and one for the performance data. As mentioned previously, we have developed two sets of both score and performance features, one for modeling piano melodies and one for modeling piano accompaniment. We describe both sets now. In the following descriptions, we denote continuous-valued features with a "c" after the

[1] `http://www.midi.org/`

[2] While MIDI can represent note loudness, it has very limited capabilities in terms of timbre manipulation, making it inappropriate for certain types of instruments. Some instruments, such as those in the wind and bowed string families, are difficult to represent adequately not only because of their complex timbral dynamics, but also because of the way in which a performer can continuously vary volume throughout the duration of a note. Because the piano does not share these abilities, MIDI provides a suitable representation.

[3] The Disklavier is a real piano outfitted with digital sensors that enable it to record performances directly to MIDI.

name, while for discrete-valued features, we add a “d”.

3.1.1 Melodic Features

The melodies used in our system are not constrained to be monophonic, as their durations are allowed to overlap. However, we do require that there is at most one note onset at each instant in time. In the interest of generality, any continuous-valued features that specify relative quantities are indifferent to tempo because they are formed from ratios rather than time-specific differences. For these features, we transform to a logarithmic domain before training and then back again before synthesis. This allows for a more perceptually plausible representation. For example, consider *RelLengthNext*, a feature describing the rhythmic relationship between a note and its successor. A note that is half as long as its successor would get $\log_2(1/2) = -1$ and a note that is twice as fast as its successor would get $\log_2(2) = 1$, thus keeping “slowing down” and “speeding up” on the same numerical scale. Additionally, we standardize the continuous feature distributions to have $\mu = 0$ and $\sigma = 1$ for training and then rescale them back for synthesis, which helps avoid certain pathologies involved in the estimation of Gaussian parameters.

3.1.1.1 Score Features

1. *FirstLast* (d): A three-valued feature that indicates whether a note begins, is internal to, or ends the piece. We found that the first and particularly last notes in performances tend to be outliers in the feature distributions. This feature helps the system to reserve special states in the HMM for these types of notes.

2. *Phrase* (c): This feature gives the relative position of each note in the phrase in which it occurs. Given that most commercial scores contain phrasing marks, this information is available to human performers. We felt that any score information available to humans ought to be available to the ESP system as well. There are several commercial software

products that allow for conversion between paper scores and symbolic representations. However, in our experiments it was a fairly easy task to extract this feature manually. If no phrase information is present in a score, there are several segmentation algorithms that could be used instead [32, 9].

3. *IntervalPrev* (c): A feature describing the interval size in semitones between this note and its predecessor. This is defined as this note's MIDI note value minus the predecessor note's MIDI note value.

4. *IntervalNext* (c): A feature describing the interval size in semitones between this note and its successor. This is defined as the successor note's MIDI note value minus this note's MIDI note value.

5. *Articulation* (c): The articulation value for this note. Articulation is defined as the inter-onset-interval (IOI) of this note divided by the duration of this note, where IOI_t = onset(note$_{t+1}$)-onset(note$_t$).

6. *RelLengthNext* (c): The duration of this note divided by the duration of the next note.

7. *RelLengthPrev* (c): The duration of the previous note divided by the duration of this note.

8. *MetricStrength* (d): An integer indicating the inherent strength of a note's position in the measure as determined by a metric hierarchy [23]. The first beat in a measure always gets the value "1" and the assignment of values to subsequent notes depends on the meter and rhythmic resolution of the measure, *e.g* four quarter notes in a 4/4 measure are assigned strengths 1, 3, 2, 3. In practice, we capped the metric strength value at 4.

3.1.1.2 Performance Features

1. *RelDuration* (c): The duration of the performed note divided by the scored note's duration. This feature provides rubato information.

2. Δ*RelDuration* (c): This note's *RelDuration* minus the previous note's *RelDuration.* This first-order difference feature is used after synthesis to smooth the transitions between states (and therefore musical contexts).

3. *Velocity* (c): The velocity with which the key on the piano corresponding to this note was struck relative to the average velocity of the piece. Velocity is a MIDI feature with values ranging from between 0 and 127. It provides a logarithmic representation of sound amplitude, generally agreeing with most theories of audio perception.

4. Δ*Velocity* (c): This note's *Velocity* minus the previous note's *Velocity.* This feature plays a similar role to that of Δ*RelDuration.*

5. *RelArticulation* (c): The articulation value for this note divided by the articulation value of the scored note.

6. Δ*RelArticulation* (c): The first-order difference feature for *RelArticulation* similar to that of Δ*Velocity.*

3.1.2 Accompaniment Features

The following features were designed for use with two-hand piano music and as such are most likely not applicable (at least in an unmodified state) to the general class of multi-voiced composition. Our approach to modeling this subset of polyphonic music is to treat chords as single units when extracting score and performance features. This representational reduction greatly simplifies the modeling problem. Additionally, the accompaniment performance features are relative to the melody performance rather than the accompaniment score. Therefore, we model the accompaniment in terms of how it follows the performed melody. The accompaniment score features are reminiscent of those used for the melody.

In a polyphonic score or performance, three basic situations can arise in each non-empty time-slice of the score or performance:

1. There is a melody event, but no accompaniment present.

2. There is an accompaniment event, but no melody present.

3. There are both melody and accompaniment events present.

These three possibilities complicate things because we are no longer guaranteed to have a value for all of the features in each event slice. We deal with this problem by maintaining three separate feature distributions at each state, one for each of the above possibilities. We will return to this subject shortly in Section 3.2.

3.1.2.1 Score Features

1. *RelLengthNextAcm* (c): The duration of this note divided by the duration of the next note.

2. *RelLengthPrevAcm* (c): The duration of the previous note divided by the duration of this note.

3. *Sonance* (c): A numerical measurement, based on a modified version of the Euler Sonance [19], that attempts to quantify the consonance/dissonance of the current chord.

   Currently, we take a somewhat simplistic approach here, as we do not take psychoacoustic influences into account. To define the Euler Sonance, first note that we can decompose a number, n, into a product of prime factors: $n = \prod_i p_i^{a_i}$ where p_i is the i^{th} prime factor and a_i is its associated exponent. Then, we define the degree of complexity of a number,

n, as: $C(n) = 1 + \sum_i a_i(p_i - 1)$. Using these definitions, we can now define the Euler Sonance of a chord with M notes having frequencies $\Phi = \{\phi_1, \phi_2, ..., \phi_M\}$ as:

$$Sonance(\Phi) = C\left(\frac{lcm(\Phi)}{gcd(\Phi)}\right) \tag{3.1}$$

Where *lcm* represents the least common multiple of its arguments and *gcd* represents the greatest common divisor of its arguments.

However, this definition is difficult to use in general because most of the note frequencies in common tuning systems are not whole integers, requiring us to estimate solutions for the *lcm* and *gcd* of real numbers. Rather than approximate, we make use of the fact that each interval associated with the chromatic scale (i.e. the interval between each note in the scale and the tonic) has an ideal small integer ratio associated with it. [4] First we find a common denominator for these ratios and scale the numerators accordingly. Then we compute the Euler Sonance of a chord by letting Φ be the newly scaled numerators corresponding to the intervals between the tonic of the key we are in and each note in the chord.

4. *MetricStrengthAcm* (d): The same as described for the melody.

3.1.2.2 Performance Features

1. *RelOnsetSlope* (c): This feature describes the onset profile (i.e. if and how a chord was "rolled") of a chord's notes relative to the performed melodic note that has the same scored onset time as that chord. If there is no melody note at the same score position, we create a pseudo-melody note by interpolating between the last and next melody notes.[5]

[4]This ratio set, from 0 to 11 semi-tones is : $\{1, \frac{16}{15}, \frac{9}{8}, \frac{6}{5}, \frac{5}{4}, \frac{4}{3}, \frac{36}{25}, \frac{3}{2}, \frac{8}{5}, \frac{5}{3}, \frac{9}{5}, \frac{15}{8}\}$

[5]Note that the pseudo-melody note is not actually added to the score, it is merely used to compute the *RelOnsetSlope* feature.

The onset profile is described by calculating the onset differences between the notes in the accompaniment chord and the performed melody note (or pseudo-melody note) and then dividing these differences by the duration of the melody note. This scales the onset deviations in an attempt to keep them appropriate for their context.

We then fit these values with a linear approximation. The approximation yields a slope and a y-intercept value (which is the "base" deviation value for the chord's profile). The slope describes how quickly the deviations from the melody note change with respect to the positions of the notes in the chord. This feature stores just the slope information obtained in this linear approximation.

2. *RelOnsetBase* (c): This feature stores the y-intercept, or "base" deviation value obtained from the approximation of the relative onset profile described above.

3. *RelDurationAcm* (c): The average duration of the notes in this chord as performed, divided by the duration of the notes as scored.

4. Δ*RelDurationAcm* (c): This chord's *RelDurationAcm* minus the previous chord's *RelDurationAcm*. This difference is used after synthesis to smooth the transitions between musical contexts in a similar fashion as Δ*RelDuration*.

5. *VelocityAcm* (c): Similar to *Velocity*, but computed as the average velocity of the notes in the chord relative to the average velocity of the piece.

6. Δ*VelocityAcm* (c): This note's *VelocityAcm* minus the previous note's *VelocityAcm*.

3.2 Representing the Data

Modeling melodic performance is relatively straightforward. The main complication is that we use both continuous and discrete-valued features and therefore must maintain a joint distribution over both types at each state. However, things are significantly more complex in the polyphonic case because we do not have a value for all of the features at every time slice. Recall from the discussion of accompaniment features that at each time slice we may have only melody features, only accompaniment features, or both types present. Our solution to this problem is to maintain separate joint discrete-continuous distributions for each possibility. Therefore, each state keeps three observation distributions, each of which is composed of a joint distribution formed from one continuous and one discrete distribution. Then, during synthesis where the score describes which of the three types of event slices are present at each step, we can just sample from the appropriate distribution.

3.2.1 Discrete-valued Features

Because the ESP system makes use of a joint distribution of discrete and continuous-valued features, we need to keep both kinds of observation distributions in each state. For the discrete features (*FirstLast* and *MetricStrength*), we use a standard multinomial distribution which we update with the entropic MAP estimator (described in Section 3.3.1). The symbols represented by this multinomial are built by taking all possible combinations of values of all of the discrete features. To build this symbol set, we just need a mapping from feature vectors to integer values. It is a fairly simple matter to construct this mapping; we need only to decide on an ordering for how we will enumerate all possible feature combinations. Using $\mathcal{F}$ to represent the set of discrete features, f_n to represent the n^{th} value of the generic discrete feature, f, O to represent the set of observations symbols, and $\mathcal{P}(x)$ to denote the powerset, the cardinality of the observation symbol set is:

$$|O| = \left|\mathcal{P}\left(\bigcup_{f\in\mathcal{F}}\bigcup_{n=1}^{|f|} f_n\right) - \{\}\right| \tag{3.2}$$

3.2.2 Continuous-valued Features and the Kernel Technique

For our continuous-valued features, we take a slightly non-traditional approach. Rather than use a standard mixture-of-Gaussians, we use a technique called Parzen windows or "kernels", adapted for use in HMMs by Wang *et al.* [35]. The idea is to build up the probability distribution at each state with a large number of small "bubbles" of probability (see Figure 3.1). These "bubbles" are essentially just Gaussians formed using data from the training set; there is one kernel per training datum and each kernel uses one training datum as its mean. Instead of learning the matrix of covariance parameters, a spherical variance of fixed size is often used. In our experiments, rather than clamping each state's set of kernel covariances to a fixed value, we adjust them after each training iteration using a suggestion by Wang *et al.* [35]:

$$\Sigma_i = \frac{variance(\mathcal{X})}{\sqrt{K_i}} \tag{3.3}$$

Here K_i represents the number of kernels used in state q_i and $\mathcal{X}$ is a set of training examples, $\mathbf{x}$, where $\mathbf{x} = \{x_1, x_2, ..., x_T\}$. In the context of the ESP system, the subscript on K_i is somewhat unncessary as we used a fixed number of kernels for all of the states, however in the interest of generality, we keep this notation.

Each of the kernels at each state corresponds to one training datum and the learning task is to find the appropriate weighting for each kernel. This is done by setting the weight of each kernel in each state equal to the probability of being in that state and observing the kernel's associated piece of data:

$$w_{i,t} = \gamma_i(t) = P(s_t = q_i | x_1, x_2, ..., x_t) \tag{3.4}$$

This is easy to do, as we have to calculate this occupancy distribution, $\gamma_i(t)$, during the E-step of training anyway. The problem, however, is efficiency. As our training data set increases in size, we have to evaluate an ever-increasing number of Gaussians, which becomes quite computationally expensive. One solution is to use a sampling-based approximation which we now discuss.

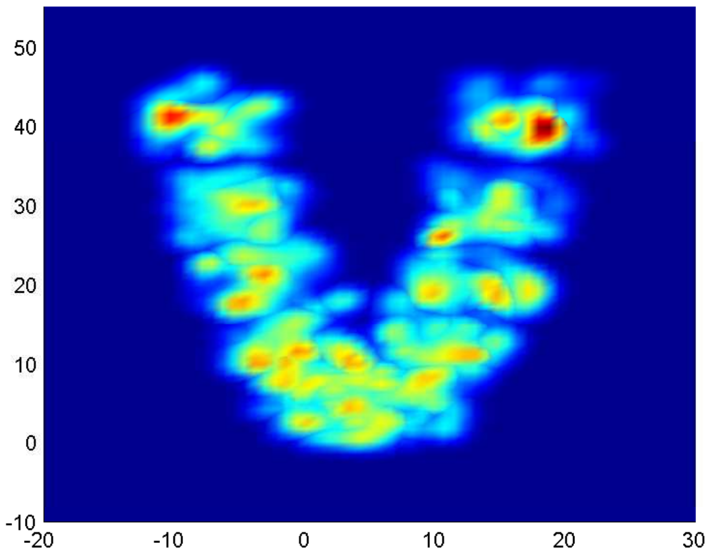

Figure 3.1: An example of a distribution represented by a set of 300 kernels. Although a mixture-of-Gaussians could be used to fit this distribution to varying degrees of accuracy, much of the fine detail would be lost.

3.2.2.1 Likelihood Weighted Sampling

One obvious way to reduce the computational burden of evaluating so many Gaussian distributions is to use a smaller fixed number of them. This means that we must decide which kernels to include in each state's set. We do this probabilistically with a technique called *likelihood weighted sampling*. In this representation, we sample the occupancy distribution in equation 3.4 to choose which kernels to include in each state's set. Note that sampling of the occupancy distribution is done with replacement and we now weight all kernels equally.

3.2.2.2 Annealing Kernels

During the early stages of training, we want to allow the model to explore the full kernel set, rather than paying too much attention to those kernels that may have received high probability from the initial parameter settings. Because we sample with replacement, it is also possible that the model will select many copies of the same kernel and specialize a state to one specific type of data point. While this may be appropriate in some circumstances, having such a spiked observation distribution often causes problems for the synthesis routine. It is likely that a new score that we wish to synthesize will contain feature values somewhat different from those contained in the training set. Therefore, this type of spiked distribution is potentially problematic because it can result in poor model generalization. To help prevent these problems, we anneal the occupancy distribution as follows:

$$w_{i,t} = \frac{\gamma_i(t)^\alpha}{\sum_{t=1}^{T} \gamma_i(t)^\alpha} \tag{3.5}$$

The temperature, α, typically follows an exponential curve from 0 to 1, which we maintain in a separate annealing schedule from that of the entropic prior (discussed in Section 3.3.3). Although, a spiked distribution is still possible because we end the schedule with $\alpha = 1$, any state in which this occurs is now more likely to be a true specialty state, reflecting a consistent type of outlier in the training examples, rather than just a training pathology.

3.3 Training

3.3.1 The Entropic MAP Estimator

We make use of a modified version of the Expectation-Maximization algorithm to train the multinomial parameters (the state transitions and discrete observations) in the HMM used in the ESP system. This modification, which replaces the maximization step of EM, is referred to as the *entropic estimator* [4]. It incorporates the belief that sparse models (those whose parameter distributions have low entropy[6]) generalize better and are therefore preferable. This bias is justified by the fact that sparse models tend to be more resistant to the effects of noise and over-fitting and therefore better able to capture the underlying structure present in the data. We can incorporate this belief into a prior distribution on model parameters, θ. This is called the *entropic prior.*

$$P_e(\theta) = e^{-H(\theta)} = e^{\sum_i \theta_i \log \theta_i} = \prod_i \theta_i^{\theta_i} = \theta^\theta \tag{3.6}$$

While classic EM tries to maximize the likelihood of the model parameters, $\mathcal{L}(\theta|\mathcal{X})$, the *entropic estimator* tries to find the maximum *a posteriori* distribution formed by combining the model likelihood with the entropic prior.[7] We can formulate this posterior using Bayes' rule as follows:

$$\begin{align} P(\theta|\mathcal{X}) &= \frac{P_e(\theta)\mathcal{L}(\theta|\mathcal{X})}{P(\mathcal{X})} \tag{3.7} \\ &= \frac{e^{-\eta H(\theta)}\mathcal{L}(\theta|\mathcal{X})}{P(\mathcal{X})} \tag{3.8} \\ &\propto e^{-\eta H(\theta)}\mathcal{L}(\theta|\mathcal{X}) \tag{3.9} \end{align}$$

$$\begin{align} \log P(\theta|\mathcal{X}) &= \log P_e(\theta) + \log \mathcal{L}(\theta|\mathcal{X}) - \log P(\mathcal{X}) \tag{3.10} \\ &= -\eta H(\theta) + \log \mathcal{L}(\theta|\mathcal{X}) - \log P(\mathcal{X}) \tag{3.11} \\ &\propto \log \mathcal{L}(\theta|\mathcal{X}) - \eta H(\theta) \tag{3.12} \end{align}$$

[6]Entropy provides a measure of information or complexity. A common form, called the *Shannon entropy* ranges from 0 to 1 and is defined as: $H(\theta) = -\sum_i \theta_i \log \theta_i$, where θ is a set of parameters. For probability distributions, it measures how certain or uncertain the distribution is. As a simple example, the uniform distribution is maximally uncertain and therefore has an entropy of 1, because all outcomes are equally likely.

[7]When $P(X|\theta)$ is referred to as a likelihood, it is a function of its second argument, which runs contrary to what the notation implies. Therefore, to avoid confusion, we adopt the notation $\mathcal{L}(\theta|X)$ to refer to the likelihood of parameters θ having generated data X.

The proportionalities of equation 3.9 and eq. 3.12 are justified because $P(\mathcal{X})$ does not depend on θ and is therefore a constant. The η variable weights the contribution of the entropic prior. When $\eta = 0$, the entropic estimator becomes equivalent to EM. When $\eta > 0$ we encourage maximum structure by rewarding entropy in an effort to model the data as fully as possible. This setting is useful in the early stages of an annealing schedule (discussed below in Section 3.3.3). When $\eta < 0$ (specifically when $\eta = -1$, see Brand [4] for details) we arrive at the minimum entropy solutions where we punish for complexity in the model. In fact, by varying η during training, typically from $\eta > 0$ down to $\eta = -1$, we can largely get around classical EM's stubborn propensity for getting stuck in local maxima (again, see Section 3.3.3 for details).

One of the key insights of the entropic MAP estimator is the idea that, once we have computed the expected sufficient statistics in the E-step of normal EM, the HMM has effectively been decoupled into a set of independent multinomials (and independent Gaussians in the case of continuous observations). This allows us to perform the maximization step on each multinomial distribution in the HMM independently. Deriving the maximization step of the entropic MAP estimator is fairly involved and so we shall not discuss it here. We provide technical details in Appendix B.

3.3.2 Trimming

Along with finding quasi-global optima, the entropic estimation framework also provides a way to roll model selection into the training process. One of the main problems with HMMs is that we must select an appropriate model to start with. We need to decide *a priori*, not only how many states, Q, the model will have, but also what the state inter-connectivity will be.[8] This is difficult if we know little about the structure of our data. If we use too many

[8]Because EM can never revive a parameter once it has zeroed out, setting transition or observation probabilities to 0 provides model structure which can be very helpful for many kinds of problems.

states, the model will over-fit the data, resulting in poor generalization. If we use too few states, we will get an overly-general model that does not fit the data well. However, by introducing trimming criteria into the training process, we can begin with an over-specified model and let the data decide what is appropriate. Granted that this is not a total solution as we still must choose an initial number of states, it is a significant move in the right direction. We now review some of the trimming tests used in the ESP system. We refer the reader to Brand's paper [4] for more extensive details.

3.3.2.1 Parameter Trimming

Because we are concerned with increasing the log-posterior in eq. 3.11 rather than just the log-likelihood, we can afford to remove a parameter if its presence increases the weighted prior (i.e. entropy) more than the log-likelihood. In practice, trimming a parameter means setting it to 0. Note that we only trim parameters when $\eta > 0$. We now present the trimming test for state transitions, although this criteria is also applicable to discrete observation symbols with essentially no change.

$$\eta H(\mathcal{A}_{i\to j}) > \mathcal{A}_{i\to j} \frac{\partial \log \mathcal{L}(\mathcal{A}_{i\to *}|\mathcal{X})}{\partial \mathcal{A}_{i\to j}} \tag{3.13}$$

$$-\eta \mathcal{A}_{i\to j} \log \mathcal{A}_{i\to j} > \tag{3.14}$$

$$\log \mathcal{A}_{i\to j} < -\frac{1}{\eta} \frac{\partial \log \mathcal{L}(\mathcal{A}_{i\to *}|\mathcal{X})}{\partial \mathcal{A}_{i\to j}} \tag{3.15}$$

$$\mathcal{A}_{i\to j} < \exp\left(-\frac{1}{\eta} \frac{\partial \log \mathcal{L}(\mathcal{A}_{i\to *}|\mathcal{X})}{\partial \mathcal{A}_{i\to j}}\right) \tag{3.16}$$

Where we have used "$*$" as a wildcard variable, meaning that the expression, $\mathcal{A}_{i\to *}$, refers to the vector of transition probabilities out of state, q_i. The inequality in eq. 3.16 provides a cheap test for trimming because we have to compute the gradient of the log-likelihood in the E step of entropic estimation already.

3.3.2.2 State Trimming

We can use similar test criteria to decide when a state may be trimmed. The following equation balances the contribution of the state q_i, with the prior probability of all transitions in and out of the state:

$$\frac{\mathcal{L}(\theta \backslash q_i|\mathcal{X})}{\mathcal{L}(\theta|\mathcal{X})} > {\mathcal{A}_{i\to i}}^{\mathcal{A}_{i\to i}} \prod_{j\neq i}^{N} {\mathcal{A}_{i\to j}}^{\mathcal{A}_{i\to j}} {\mathcal{A}_{j\to i}}^{\mathcal{A}_{j\to i}} \tag{3.17}$$

However, this inequality is significantly more computationally expensive to implement than that of the parameter trim test because we have to compute the likelihood of the model without each state we wish to test for trimming. Brand [4] suggests the following heuristic approximation instead:

1. Use the above transition trimming criteria, but bias the trimming to self-transitions first.
2. Compute the occupancy probability of each state (the number of expected visits to the state divided by the total amount of data).
3. Trim any state with an occupancy probability less than some threshold. In practice, we use a simple function of the number of states in the model.

3.3.3 Annealing

As mentioned above, traditional EM suffers from the tendency to get stuck in local maxima of the probability space defined by the model parameters.[9] The degree to which this happens is largely a function of the initial parameter settings we assign to the model. There are heuristics which can be used to help alleviate this problem, such as k-means clustering to initialize Gaussian parameters (which we do in our model), however they are not a total solution. Annealing is a concept which comes out of metallurgy and crystallography. The idea is that,

[9]This space is also referred to as the *energy surface*, a term born out of simulated annealing's origins in the physics world. We refer to both in our discussions below.

by slowly lowering the temperature of a piece of metal, its molecules will have time to settle into a structurally optimal configuration. This same principle can be applied to optimization problems.

The intuition behind annealing, is that at high "temperatures" (negative values of η in the case of the entropic MAP estimator), we are smoothing the energy surface of the function which we are trying to maximize. This gives the algorithm a better chance of getting into the right "neighborhood" of the function's global maximum. Figure 3.2 shows how a probability distribution gets flattened out with a high temperature and then regains detail as the temperature is lowered. As the temperature approaches ∞, the distribution approaches uniform, while when the temperature is 0, the distribution is unchanged from its original value.

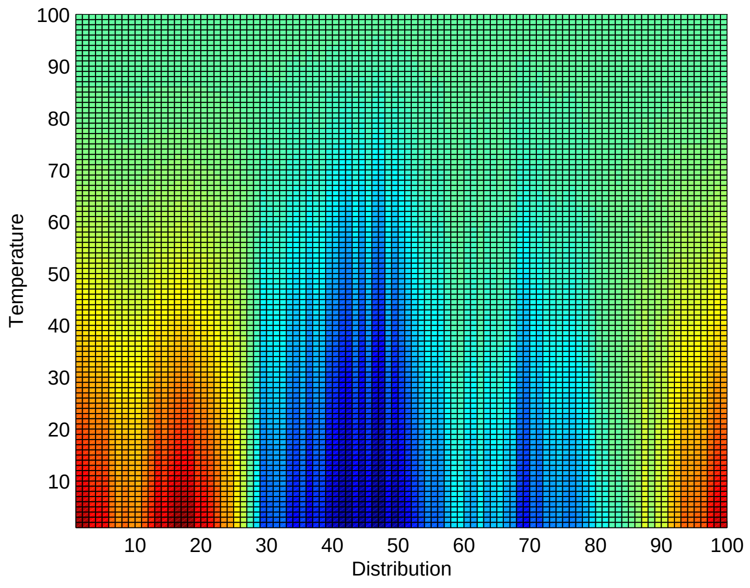

Figure 3.2: Here we show the energy surface of a probability distribution and how it varies as we anneal it. Green represents the value of the uniform distribution (which in this plot is .01), red indicates a value higher than that of the uniform, and blue indicates a lower value.

We employ three different types of annealing in the ESP system: annealing the contribution of the entropic prior, annealing the state occupancy distributions used to select kernels, and annealing the kernel contribution distribution used in the synthesis routine. We discuss the first of these three now.

3.3.3.1 Annealing the Entropic Prior

We can anneal the posterior distribution, which the entropic MAP estimator tries to maximize, by changing the prior weighting parameter, η. To do this we vary η continuously over the course of all training iterations, typically starting at around 2 and ending at -1. There is little mathematical theory dealing with how exactly to choose the annealing schedule. However, as mentioned previously, $\eta = 0$ and $\eta = -1$, are special cases, which correspond to EM and the minimum entropy estimator, respectively. Depending on our goals for training, we might choose to stop at $\eta = 0$ (if we were more concerned with model fit than generalization) instead of allowing the algorithm to seek the minimum entropy solution.

During the first few iterations of training, we want to give the algorithm a chance to explore the target function's energy surface at a larger scale. Then as we heat up the temperature, more detail comes into view, allowing us to settle into a (hopefully) global maximum.

3.4 Synthesis

Once we have trained our model using the above techniques, we can use it to synthesize renditions of novel scores. We do this by first extracting the features from the new score in the same way as with our training data. The next step is to find the Viterbi path [34], $\mathbf{v} = \{v_1, v_2, ..., v_{|newscore|}\}$, through the model using the marginal observation distribution of score features and the new score data. [10] This is done by first computing the score's marginal observation distribution at each state, a relatively simple matter for Gaussians as it involves just keeping those dimensions of the Gaussian that correspond to score features. For the discrete features, we don't need to change the distribution as all of the performance features are continuous-valued.

Next, we compute an initial sequence of rendition features for the score by sampling the distribution at each state in the Viterbi path, $\mathbf{v}$, formed by conditioning on the new score features. Denoting the score feature indices as, S, and the rendition feature indices as, R, the standard formulas for a Gaussian with $\mu = [RS]^T$ and $\Sigma = \begin{bmatrix} RR & RS \\ SR & SS \end{bmatrix}$, conditioned on some data, x_S, are:

$$\mu_{R|S} = \mu^R + \Sigma^{RS}(\Sigma^{RR})^{-1}(x_S - \mu^S) \tag{3.18}$$

$$\Sigma_{R|S} = \Sigma^{RR} - \Sigma^{RS}(\Sigma^{SS})^{-1}\Sigma^{SR} \tag{3.19}$$

After we have an initial sequence of rendition features, we use the EM algorithm to iteratively update the rendition sequence until we have found the rendition feature sequence that, in combination with the new score feature sequence, has the highest probability given the model parameters. The EM steps proceed in the usual way, first computing the expected sufficient statistics needed to update our current estimate of the renditon feature sequence, and then maximizing. In our case, the E-step statistics are composed of a probability distribution, λ, giving the probability of observing a data sequence, $\mathbf{x}$, at each time step, using each kernel, in each state:

[10]The Viterbi algorithm uses dynamic programming to find the sequence of states through the HMM that maximizes the probability of a particular observation sequence.

$$\lambda_t(i,k) = P(s_t = q_i, \mu_{i,k} = x_t) \tag{3.20}$$

$$= \gamma_t(i) \frac{\mathcal{N}(x_t|\mu_{i,k}, \Sigma_i)}{\sum_{j=1}^{K_i} \mathcal{N}(x_t|\mu_{i,j}, \Sigma_i)} \tag{3.21}$$

Here $\mu_{i,k}$ refers to the k^{th} kernel in state q_i and K_i refers to the number of kernels in state q_i.

The maximization step uses the following formula, which is derived in the usual EM way (i.e. taking derivatives of an auxiliary function which approximates the target function, setting to 0, and solving). See Wang *et al.* for details [35].

$$R_t = \sum_{i=1}^{Q} \sum_{k=1}^{K_i} \lambda_t(i,k)\, \mu_{q,k}^{R}\, \Sigma_q^{RR} \left[\sum_{i=1}^{Q} \sum_{k=1}^{K_i} \lambda_t(i,k)\, \Sigma_q^{RR} \right]^{-1} \tag{3.22}$$

Once we have synthesized our new rendition features, we transform any features in logarithmic space back, re-scale the features to have the values of μ and σ that the training performance features originally had, and finally reverse the feature transformations to obtain a MIDI file.

3.4.0.2 Annealing Kernel Contributions

After we have run EM to convergence, we are left with a sequence of rendition features which, in combination with the new score features, have the (hopefully) highest probability of being generated by the model. However, because EM makes no guarantees about global optimality, there is still the possibility that we have found a sub-optimal rendition for the new score. We can reduce the chance of this happening by annealing the distribution of kernel contributions, $\lambda_t(i,k)$.

We anneal $\lambda_t(i,k)$ in a fashion similar to the occupancy distribution, $\gamma_i(t)$, used in selecting kernels during training. That is, we use the following modification to $\lambda_t(i,k)$:

$$\lambda_t(i,k) = \frac{\left(\gamma_t(i)\frac{\mathcal{N}(x_t|\mu_{i,k},\Sigma_i)}{\sum_{j=1}^{K_i}\mathcal{N}(x_t|\mu_{i,j},\Sigma_i)}\right)^{\alpha}}{\sum_{t=1}^{T}\left(\gamma_t(i)\frac{\mathcal{N}(x_t|\mu_{i,k},\Sigma_i)}{\sum_{j=1}^{K_i}\mathcal{N}(x_t|\mu_{i,j},\Sigma_i)}\right)^{\alpha}} \tag{3.23}$$

Generally, we have found that temperature schedules similar to that used to anneal $w_{i,t}$, yield acceptable results for $\lambda_t(i,k)$.

3.4.0.3 Feature Smoothing

A potential problem of using a model with a finite number of discrete states, is that we may end up with abrupt changes in musical context. This can lead to an unpleasant jerkiness in a synthesized rendition's volume and rubato. To alleviate this potential problem, we use the newly synthesized first-order difference rendition features to smooth those rendition features that are used in conversion from feature space to audio (MIDI).

This is done by forming a block-tridiagonal system of linear equations which can be easily solved to yield the smoothed feature set. Below, we refer to the index of the feature to be smoothed as f, and the index of its corresponding first-order difference feature as Δf.

$$A_t = \sum_{i=1}^{Q}\sum_{k}^{K_i}\lambda_t(i,k)\mu_{i,k}^{f}\Sigma_i^{f} \tag{3.24}$$

$$B_t = \sum_{i=1}^{Q}\sum_{k}^{K_i}\lambda_t(i,k)\mu_{i,k}^{\Delta f}\Sigma_i^{\Delta f} \tag{3.25}$$

$$C_t = \sum_{i=1}^{Q}\sum_{k}^{K_i}\lambda_t(i,k)\Sigma_i^{f} \tag{3.26}$$

$$D_t = \sum_{i=1}^{Q}\sum_{k}^{K_i}\lambda_t(i,k)\Sigma_i^{\Delta f} \tag{3.27}$$

$$R_f = \begin{bmatrix} C_1 + D_1 + D_2 & -D_2 & 0 & \cdots & \cdots & 0 \\ -D_2 & C_2 + D_2 + D_3 & -D_3 & 0 & \cdots & 0 \\ 0 & -D_3 & \ddots & \cdots & \cdots & 0 \\ \vdots & 0 & \vdots & \ddots & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & -D_{|R|} \\ 0 & 0 & 0 & 0 & -D_{|R|} & C_{|R|} + 2D_{|R|} \end{bmatrix} \tag{3.28}$$

Chapter 4

Experiments

The ESP system was implemented entirely in the Matlab programming language, making use of the BayesNet toolbox[1] for some machine learning and statistical routines and the Miditoolbox library[2] for MIDI-related I/O tasks. To test the quality of the renditions generated by the ESP system, we conducted a series of informal listening experiments. First, we trained the ESP system, using 30 states and 200 kernels per state, on 10 performances of the melody of Schumann's *Traumerei* performed by graduate music students[3]. The system was then used to generate an expressive rendition of a different piece, the first 10 bars of the melody of Chopin's Etude Op. 10 No. 3. We also created three other 10-bar renditions of the Chopin Etude: a literal rendition of the score (i.e. a performance with no expressive timing deviations and constant velocity), a performance by a local graduate student studying piano performance, and a performance by an advanced undergraduate majoring in piano performance. Each of the human performers was given the score in advance and was also allowed a warm-up run before recorded three takes of the Chopin Etude. We then used the best sounding take (to our ears) in the

[1]`http://www.ai.mit.edu/~murphyk/Software/BNT/bnt.html`
[2]`http://www.jyu.fi/musica/miditoolbox/`
[3]We are deeply indebted to Bruno Repp at Yale for making this data available.

experiment.

For the listening test, we asked fourteen local music students to listen to and rank three performances: the expressive synthetic rendition, the literal synthetic rendition, and one of the human performances. The decision to enlist students specifically studying music was motivated by the belief that musically educated participants should be better able to assess performance quality. The performances were presented in random order, and the listener could re-hear performances as often as desired. The results are summarized in the following table.

	Literal	Ugrad+Grad	Expressive
best	2	1 + 2	9
middle	5	3 + 5	1
worst	7	3 + 0	4

Although far from conclusive, this data does show some interesting trends. First, although some listeners (4 of 14) preferred the inexpressive score to the expressive synthesis, most (10 of 14) ranked the expressive synthesis higher. The expressive synthesis also was preferred over the undergraduate's performance (5 of 7 times). The expressive synthesis seemed comparable (preferred 4 of 7 times) to that of the graduate student, who presumably has a similar skill level as the students who produced the training data. Although, the sample size is small, these results do give modest evidence that our system is successfully able to synthesize performances of novel scores at a skill level approximating that of the performers who generated the training data.

Chapter 5

Conclusion

We have presented a machine learning approach to the difficult problem of modeling and synthesizing expressive musical renditions. Our approach is efficient and applicable to any instrument or musical style for which an appropriate set of features can be extracted. Through informal listening tests, we have demonstrated that our system is successful at capturing the general trends in timing deviation and loudness change present in the example performances of skilled human pianists. Although our results are certainly not conclusive, we believe they provide evidence of the viability of our approach to this problem.

5.0.0.4 Applications

The ESP system has practical applications as a tool for "bringing life" to raw musical scores which tend to be somewhat emotionally vapid when rendered literally. Such a system could be useful to composers, who may find the ability to hear a piece as it might sound when performed, helpful during the composition process. As a result of the entropic pruning technique, our system produces HMMs which are sparse (i.e. have relatively low state interconnectivity) and therefore are more interpretable than what classical training methods might yield. We believe that by examining the structure of a trained model, it may be possible to gain new musicological insight into the relationship between performance and compositional structure.

5.0.0.5 Future Work

We are currently working on several improvements to our system. First, we are working to grow our set of training data. We feel that this is a particularly important task as it will likely lead to more general models by reducing the influence of any one score-performance data subset. Second, we would like to expand the existing score and performance features (both for piano and other instruments) to include information on multiple time scales. Music is an extremely hierarchical phenomenon and one of the problems that the system currently has is in synthesizing longer melodies. Although HMMs (particularly sparse models) are capable of carrying contextual information structually, it may be beneficial to add features which explicitly represent score and/or performance information on a time scale beyond note-to-note. One way to do this may be to use theories of musical cognition, such as Narmour's Implication-Realization theory [24, 25] and Lerdahl and Jackendoff's Generative Theory of Tonal Music [23], to generate higher level score features. Another possibility, which may complement higher-level features, is to use a hierarchical state-space model. Hierarchical hidden Markov models [18], provide a way to factor the state space by allowing states to call sub-HMMs and therefore output strings of observations. This can potentially lead to a much more efficient representation because sub-HMMs can be shared by several higher level states.

Our system currently builds a probabilistic model $P(R, S|\theta) = P(R|S, \theta)P(S|\theta)$ from the data. Thus far we have exploited the distribution $P(R|S, \theta)$ to create synthetic renditions of new scores. There are several interesting ways in which we might be able to use our model for musical classification and/or information retrieval tasks. When there are multiple data sets, perhaps representing different composers, performers, or styles, we can learn a different model θ_i

for each data set. Scores (or performances) can then be classified by finding the θ_i maximizing $P(S|\theta_i)$ (or $P(R|S, \theta_i)$). One way to measure the similarity between two performances is to train a model using one of them and then measure the probability of the other given the model. To measure the similarity between models of two or more data sets we could use a divergence function, such as Juang and Rabiner's cross-model entropy [22]. Going even further, one might create a set of n reference θ_i's and use (some function of) the values $P(S|\theta_i)$ (or $P(R|S, \theta_i)$) to map scores (or performances) into R^n.

In summary, we strongly believe that hidden Markov models and graphical models in general, have a significant role to play in the future of computational musicology. Aside from finding new ways in which to exploit these particular types of statistical learning frameworks, there appears to be many ways in which the area would benefit from different algorithmic approaches as well. It is our sincere hope that the ideas presented herein will not only convince the reader of the validity of our approach, but will also inspire others to pursue this fascinating problem domain.

Appendix A

Hidden Markov Models

A.1 Introduction

A hidden Markov model describes a probability distribution that varies over time. This non-stationary distribution can be represented by the following parameterization: $\theta = \{\pi, \mathcal{A}, \mathcal{B}\}$. These parameters describe a graph containing a set of nodes or states, $\mathcal{Q} = \{q_1, q_2, ..., q_N\}$. Each state is connected to each other state by a transitional arc which has a probability associated with it. $\mathcal{A}$ represents the matrix of such probabilities where each entry, $A_{i \to j}$, gives the probability of moving from state q_i to state q_j. The transitions out of each state form a multinomial [1] probability distribution and are therefore stochastic, meaning that they sum to 1 (i.e. $\sum_{j=1}^{|N|} A_{i \to j} = 1$). $\mathcal{B} = \{B_1, B_2, ..., B_N\}$ is a set of local probability distributions (one per state) which describes how each state emits "observations" or output values. It is important to note that at each time step we don't know which state has emitted the current observation; this value is "hidden" from us and must be inferred probabilistically. Finally, π is a distribution which describes how likely it is that we begin in each state.

[1] Recall that a multinomial is defined as: $P(\omega|\lambda) = \prod_i {\lambda_i}^{\omega_i}$, where λ is a set of parameters and ω is *evidence*, which is in the form of usage counts.

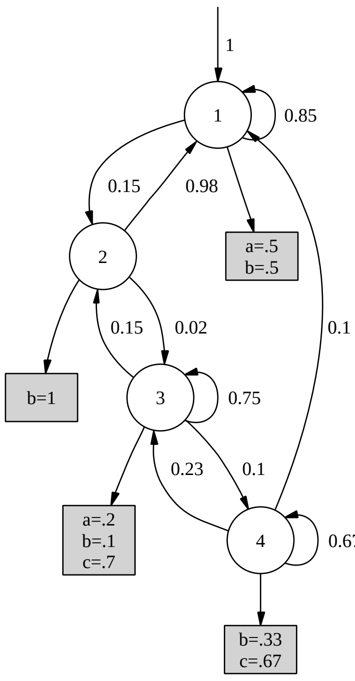

Figure A.1: A simple 4-state HMM that outputs three types of symbols: "a", "b", and "c". Clear circular nodes represent hidden states while shaded square nodes represent observables. Transitions and observations are labeled with their probabilities and the arrow at the top gives the prior probability of starting in each state.

Intuitively, an HMM operates as follows: first we sample π to decide which state to begin in. Then at each of the following time steps, we first output an observation, according to the local probability distribution, B_i, and then make a probabilistic move to a new state by sampling the current state's outgoing transition probabilies, $A_{i\rightarrow *}$. Therefore, the probability of a data sequence, $\mathcal{X}$, given a sequence of states, $\mathbf{s}$ and an HMM with parameters $\theta = \{\pi, \mathcal{A}, \mathcal{B}\}$ is the product of the first state's start probability, the transitions probabilities along the path, and the observations probabilities along the path:

$$P(\mathbf{x}|\mathbf{s}, \theta) = \pi_{s_1} B_{s_1}(\mathbf{x}) \prod_{t=2}^{|\mathbf{s}|} A_{s_{t-1}\rightarrow s_t} B_t \tag{A.1}$$

This quantity is the *complete-data* likelihood because it assumes that we were given the path through the HMM as well as the data. When we sum this quantity over all possible state sequences, $\mathcal{S}$, we arrive at the *incomplete-data* likelihood, which is the probability of a data sequence having been generated by a particular HMM:

$$P(\mathbf{x}|\theta) = \sum_{\mathbf{s} \in \mathcal{S}} \pi_{s_1} B_{s_1}(\mathbf{x}) \prod_{t=2}^{|S|} A_{s_{t-1} \to s_t} B_t \tag{A.2}$$

In practice, we usually take logarithms of these quantities because the multiplication of probabilities falls below machine precision after only a small number of iterations. Thus, eq. (A.1) becomes the *complete-data* log-likelihood:

$$\log P(\mathbf{x}|\mathbf{s}, \theta) = \log \pi_{s_1} + \log B_{s_1}(\mathbf{x}) + \sum_{t=2}^{|\mathbf{s}|} \log A_{s_{t-1} \to s_t} + \sum_{t=2}^{|\mathbf{s}|} \log B_t \tag{A.3}$$

Note that when we take the logarithm of eq. (A.2), we obtain a quantity that is extremely difficult to optimize because we have the logarithm of a sum. We will return to this issue in Section A.3

A.2 Observations

There are two main classes of hidden Markov models: those with discrete-valued observations and those with continuous-valued observations. In the case of discrete observations, each B_i is a multinomial and has an associated alphabet of output symbols, $\mathcal{O} = \{o_1, o_2, ..., o_M\}$.

For continuous observations, one popular choice of distribution is the Gaussian. Recall that, given mean, μ, and covariance, Σ, the Gaussian distribution for state q_i is defined as follows:

$$\mathcal{N}_i(\mathbf{x}|\mu_i, \Sigma_i) = \frac{e^{-(\mathbf{x}-\mu_i)^T \Sigma_i^{-1} (\mathbf{x}-\mu_i)/2}}{\sqrt{(2\pi)^d |\Sigma_i|}} \tag{A.4}$$

In practice, we are often modeling data too complex to be described by a single Gaussian at each state. In this case, we can use a mixture, $\mathcal{N} = \{\mathcal{N}_1, \mathcal{N}_2, ..., \mathcal{N}_M\}$, of M Gaussians instead. This introduces an additional mixing parameter, $\mathbf{w} = \{w_1, w_2, ..., w_M\}$, which gives the probabilities of each of the component Gaussians in a state being used to generate observations. Therefore, each state's observation distribution parameters are: $B_i = \{\mathbf{w}, \mathcal{N}\}$.

A.3 Training

Usually, we are given a set of data sequences $\mathcal{X}$, where each sequence, $\mathbf{x} \in \mathcal{X}$, is comprised of a set of observations such that $\mathbf{x} = \{x_1, x_2, ..., x_T\}$. For an HMM with discrete observations, each x_i corresponds to a single symbol, while in an HMM with Gaussian observations of dimension D, each x_i corresponds to a single D-dimensional observation.

In order to model some data with an HMM, we must first decide how many states to use. This is not always an easy decision to make (especially when we know little about the complexity of our data), making model selection one of the classic difficulties of HMMs.[2] Once we have chosen a number of states as well as some initial settings for the model parameters (which are often just random), we would like to adjust these parameters so that the log-likelihood of the model given the data is maximized. The difficulty, is that we don't have all of the information necessary to optimize the parameters directly because the state sequence is *hidden.* Traditionally, we get around this using a dynamic-programming technique, called the Baum-Welch algorithm [2]. The Baum-Welch algorithm, which is a variant of the Expectation-Maximization algorithm [15], lower-bounds the *incomplete-data* log-likelihood by pretending that we do know the state sequence and then maximizing the *complete-data* log-likelihood instead. When we treat $\mathbf{s}$ as a random variable rather than a given value, the complete-data log-likelihood becomes:

[2]We discuss some techniques to aid in this problem in Chapter 3.3.1

$$\log P(\mathbf{x}, \mathbf{s}|\theta) = \log \pi_{s_1} + \log B_{s_1}(\mathbf{x}) + \sum_{t=2}^{|\mathbf{s}|} \log A_{s_{t-1} \to s_t} + \sum_{t=2}^{|\mathbf{s}|} \log B_t \tag{A.5}$$

The basic idea behind Expectation-Maximization is to use our "best guess" of the parameters at the current time step to first compute the expected value of the incomplete-data log-likelihood with respect to **s**, and then maximize this expectation to arrive at a new "best guess". This two-step process repeats until the log-likelihood converges.

In the case of HMMs, these steps are particularly intuitive. The expectation step takes the form of finding the expected number of times each parameter in the model is used. To maximize these expectations, we only need to normalize to ensure our distributions are stochastic.

Baum-Welch uses a dynamic-programming recurrence to compute the expected parameter usages efficiently: Let $\alpha_i(t)$ be the probability of being in state q_i after having observed the partial data sequence up to time t. We can compute this quantity recursively as follows:

1. $\alpha_i(1) = \pi_i B_i(x_1)$
2. $\alpha_i(t+1) = \left(\sum_{i=1}^{N} \alpha_i(t) A_{i \to j}\right) B_j(x_{t+1})$
3. $P(\mathbf{x}|\theta) = \sum_{i=1}^{N} \alpha_i(T)$

This quantity is referred to as the "forward" pass. There is an equivalent "backward" pass as well which defines a variable, $\beta_i(t)$, giving the probability of transitioning out of state, q_i, and observing the data sequence from time t onwards.

1. $\beta i(T) = 1$
2. $\beta i(t) = \sum_{j=1}^{N} A_{i \to j} B_j(x_{t+1}) \beta_j(t+1)$
3. $P(\mathbf{x}|\theta) = \sum_{i=1}^{N} \beta_i(1) \pi_i B_i(x_1)$

Together these variables can be used to compute a number of expectations. First, let us define, $\gamma_i(t)$, which is the state the occupancy variable. It gives the probability of being in state q_i at time t of a data sequence.

$$\gamma_i(t) = \frac{\alpha_i(t)\beta_i(t)}{\sum_{j=1}^{N} \alpha_j(t)\beta_j(t)} \tag{A.6}$$

We can also define, $\xi_{i,j}(t)$, which gives the probability of transitioning from state q_i to state q_j at time t of a data sequence:

$$\xi_{i,j}(t) = \frac{\gamma_i(t)A_{i\to j}B_j(x_{t+1})\beta_j(t+1)}{\beta_i(t)} \tag{A.7}$$

With these variables in hand, we are now in a position to compute the expected parameter usages. First, we can compute the expected number of times that we will visit each state, q_i, by summing $\gamma_i(t)$ over the whole data sequence: $\sum_{t=1}^{T} \gamma_i(t)$. We can do the same with $\xi_{i,j}(t)$ to obtain the expected number of transitions from each state to each other: $\sum_{t=1}^{T-1} \xi_{i,j}(t)$.

Computing the new values of the start and transition parameters in the maximization step is now fairly straightforward:

$$\hat{\pi}_i = \gamma_i(1) \tag{A.8}$$

$$\hat{A}_{i\to j} = \frac{\sum_{t=1}^{T-1} \xi_{i,j}(t)}{\sum_{t=1}^{T-1} \gamma_i(t)} \tag{A.9}$$

The observation parameters are very similar to the above for discrete case:

$$\hat{B}_i(k) = \frac{\sum_{t=1}^{T} \delta(o_t, v_k)\gamma_i(t)}{\sum_{t=1}^{T} \gamma_i(t)} \tag{A.10}$$

Where $\delta(o_t, v_k)$ is the Kronecker delta function which is 1 when its arguments are equal and 0 otherwise.

For Gaussian observation HMMs, we will need an additional auxiliary variable: $\gamma_{i,m}(t)$. This variable gives the probability that the observation at time t was generated in state q_i by the state's m^{th} Gaussian. Below we refer to the m_{th} Gaussian in state q_i as $B_{i,m}$.

$$\gamma_{i,m}(t) = \gamma_i(t) \frac{w_{i,m} B_{i,m}(x_t)}{B_i(x_t)} \tag{A.11}$$

Now we can compute the Gaussian parameters:

$$\hat{w}_{i,m} = \frac{\sum_{t=1}^{T} \gamma_{i,m}(t)}{\sum_{t=1}^{T} \gamma_i(t)} \tag{A.12}$$

$$\hat{\mu}_{i,m} = \frac{\sum_{t=1}^{T} \gamma_{i,m}(t) x_t}{\sum_{t=1}^{T} \gamma_{i,m}(t)} \tag{A.13}$$

$$\hat{\Sigma}_{i,m} = \frac{\sum_{t=1}^{T} \gamma_{i,m}(t)(x_t - \mu_{i,m})(x_t - \mu_{i,m})^T}{\sum_{t=1}^{T} \gamma_{i,m}(t)} \tag{A.14}$$

Appendix B

Deriving the Entropic MAP Estimator

In the following derivations, we refer to the expected sufficient statistics (i.e. usage counts gathered in the E-step) for a multinomial distribution, θ, as the *evidence*, ω. Given evidence ω, the posterior for a multinomial distribution is:

$$\begin{aligned} P(\theta|\omega) &\propto P_e(\theta)\mathcal{L}(\theta|\omega) \\ &= \prod_i \theta_i^{\eta\theta_i} \prod_i \theta_i^{\omega_i} \\ &= \prod_i \theta_i^{\eta\theta_i+\omega_i} \end{aligned} \tag{B.1}$$

To maximize parameter θ_i, we add a Lagrange multiplier to ensure that each multinomial sums to 1, take derivatives of the log-posterior, set them 0, and solve:

$$
\begin{aligned}
0 &= \frac{\partial}{\partial \theta_i}\left[\log\left(\prod_i \theta_i^{\eta\theta_i+\omega_i}\right)+\lambda\left(\sum_i \theta_i - 1\right)\right] \\
&= \frac{\partial}{\partial \theta_i}\left[\sum_i \log\left(\theta_i^{\eta\theta_i}\theta_i^{\omega_i}\right)+\lambda\left(\sum_i \theta_i - 1\right)\right] \\
&= \frac{\partial}{\partial \theta_i}\left[\sum_i \left(\log\left(\theta_i^{\eta\theta_i}\right)+\log\left(\theta_i^{\omega_i}\right)\right)+\lambda\left(\sum_i \theta_i - 1\right)\right] \\
&= \frac{\partial}{\partial \theta_i}\left[\sum_i \left(\eta\theta_i \log\theta_i + \omega_i \log\theta_i\right)+\lambda\left(\sum_i \tilde{\theta}_i - 1\right)\right] \\
&= \frac{\partial}{\partial \theta_i}\left[\sum_i \left((\eta\theta_i + \omega_i)\log\theta_i\right)+\lambda\left(\sum_i \theta_i - 1\right)\right] \\
&= \frac{\omega_i}{\theta_i} + \eta\log\theta_i + \eta + \lambda \qquad \text{(B.2)}
\end{aligned}
$$

We can easily solve (B.2) for λ:

$$
\lambda = -\frac{\omega_i}{\theta_i} - \eta\log\theta_i - \eta \qquad \text{(B.3)}
$$

Solving for θ_i is a bit more tricky due to the mixed polynomial and logarithmic terms. However, we can do so by making use of the Lambert W function, which satisfies the following equality: $W(y)e^{W(y)} = y$ (see Figure B.1). There are a variety of techniques for efficient calculation of the W function. For details, see [5].

Now we show how to work backwards from the W function and arrive at eq. (B.2). To begin, note that the W function can be re-written as: $W(y) + \log(W(y)) = \log(y)$. Now let $y = e^m$.

$$
\begin{aligned}
0 &= -W(e^m) - \log(W(e^m)) + m \\
&= \frac{-z}{z/W(e^m)} - \log(W(e^m)) + m + \log z - \log z \\
&= \frac{-z}{z/W(e^m)} + \log\left(\frac{z}{W(e^m)}\right) + m - \log z \qquad \text{(B.4)}
\end{aligned}
$$

Now, let $m = 1 + \frac{\lambda}{\eta} + \log z$. Substituting in for m and continuing, we have:

$$\begin{aligned}
0 &= \frac{-z}{z/W\left(e^{1+\frac{\lambda}{\eta}+\log z}\right)} + \log\left(\frac{z}{W\left(e^{1+\frac{\lambda}{\eta}+\log z}\right)}\right) + 1 + \frac{\lambda}{\eta} \\
&= \frac{-z}{z/W\left(ze^{1+\frac{\lambda}{z}}\right)} + \log\left(\frac{z}{W\left(ze^{1+\frac{\lambda}{z}}\right)}\right) + 1 + \frac{\lambda}{\eta} \\
&= \frac{\frac{\omega_i}{\eta}}{-\frac{\omega_i}{\eta}/W\left(-\frac{\omega_i}{\eta}e^{1+\frac{\lambda}{\eta}}\right)} + \log\left(\frac{-\frac{\omega_i}{\eta}}{W\left(-\frac{\omega_i}{\eta}e^{1+\frac{\lambda}{\eta}}\right)}\right) + 1 + \frac{\lambda}{\eta} \qquad \text{(B.5)}
\end{aligned}$$

Where we have let $z = -\frac{\omega_i}{\eta}$. This implies that:

$$\theta_i = -\frac{\omega_i}{\eta W\left(-\frac{\omega_i}{\eta}e^{1+\lambda}\right)} \qquad \text{(B.6)}$$

Substituting (B.6) into (B.5), we get:

$$0 = \frac{\omega_i}{\eta\theta_i} + \log\theta_i + 1 + \frac{\lambda}{\eta} \qquad \text{(B.7)}$$

Which is the derivative of the log-posterior (B.2) divided by η. Equation (B.6) and equation (B.2) form a fix-point solution for λ and therefore θ_i. We can iteratively update the *current update's* estimate of θ by first solving for θ given λ, normalizing θ, and then calculating λ given θ. Convergence is fast (2-5 iterations). Brand [5] provides ideas for how to initialize λ.

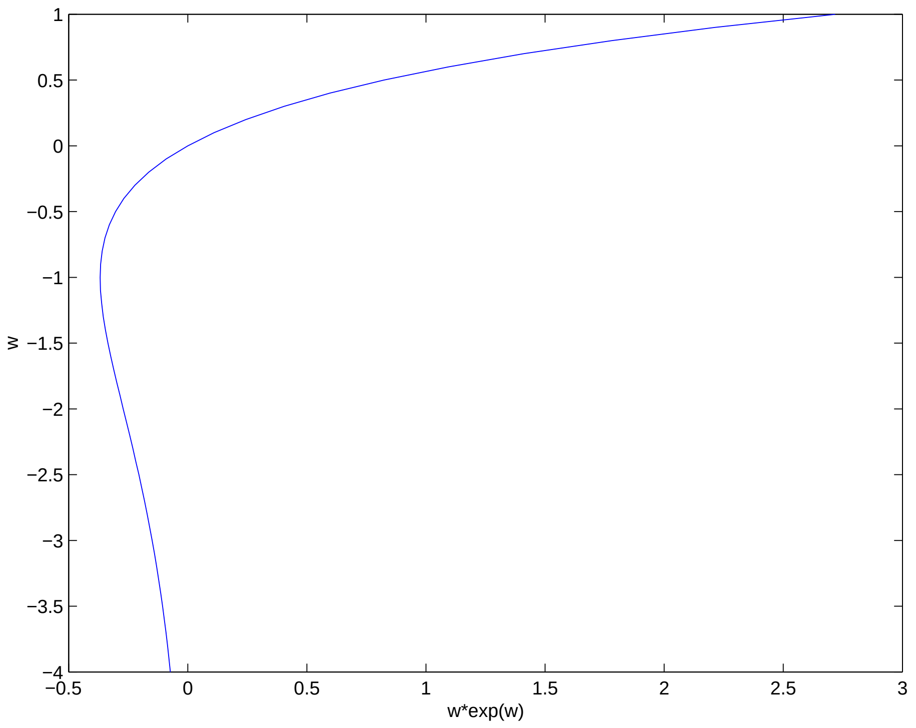

Figure B.1: This plot shows the real-valued branches of the Lambert W function. The W function is the multivalued inverse of $w \to we^w$ and so we can plot the real branches indirectly, by plotting $w \to we^w$ with the axes swapped.

Bibliography

[1] J.L. Arcos and R. López de Mántaras. An interactive cbr approach for generating expressive music. *Journal of Applied Intelligence*, 27(1):115–129, 2001.

[2] L.E. Baum. An inequality and associated maximization technique in statistical estimation for probabilistic functions of markov processes. *Inequalities*, 3:1–8, 1972.

[3] J. Bloch and R. Dannenberg. Real-time computer accompaniment of keyboard performances. In B. Truax, editor, *Proceedings of the 1985 International Computer Music Conference*. International Computer Music Association, 1985.

[4] M. Brand. Pattern discovery via entropy minimization. In D. Heckerman and C. Whittaker, editors, *Artificial Intelligence and Statistics*. Morgan Kaufman, January 1999.

[5] M. Brand. Voice puppetry. In A. Rockwood, editor, *Siggraph 1999, Computer Graphics Proceedings*, pages 21–28, Los Angeles, 1999. Addison Wesley Longman.

[6] M. Brand. Style machines. In *Siggraph 2000, Computer Graphics Proceedings*, 2000.

[7] R. Bresin. Articulation rules for automatic music performance. In *Proceedings of the International Computer Music Conference*, 2001.

[8] R. Bresin, A. Friberg, and J. Sundberg. Director musices: The kth performance rules system. In *Proceedings of SIGMUS-46*, 2002.

[9] E. Cambouropoulos. The local boundary detection model (lbdm) and its application in the study of expressive timing. In *Proceedings of the International Computer Music Conference*, 2001.

[10] M. Casey. Musical structure and content repurposing with bayesian models. In *Proceedings of the Cambridge Music Processing Colloquium*. University of Cambridge, 2003.

[11] A.T. Cemgil and H.J. Kappen. Monte carlo methods for tempo tracking and rhythm quantization. *Journal of Artifical Intelligence Research*, 18:45–81, 2003.

[12] A.T. Cemgil, H.J. Kappen, and D. Barber. Generative model based polyphonic music transcription. In D. Heckerman and C. Whittaker, editors, *IEEE workshop on applications of signal processing to audio and acoustics*, New Paltz, NY, October 2003.

[13] D. Cope. *Experiments in Musical Intelligence.* A-R Editions, 1996.

[14] D. Cope. *Virtual Music: Computer Synthesis of Musical Style.* MIT Press, 2001.

[15] A.P. Dempster, N.M. Laird, and D. Rubin. Maximum-likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistics Society, Series B*, 39(1):1–38, 1977.

[16] S. Dixon. Automatic extraction of tempo and beat from expressive performances, 2001.

[17] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* Wiley, 1973.

[18] S. Fine, Y. Singer, and N. Tishby. The hierarchical hidden markov model: Analysis and applications. *Machine Learning*, 32(1):41–62, July 1998.

[19] F.J. Sánchez González. Two numerical theories of sonance. **url**:http://www.uam.es/personal_pdi/filoyletras/jsango/appendix_two%20numerical.htm.

[20] A. Friberg. Generative rules for music performance: A formal description of a rule system. *Computer Music Journal*, 15(2):56–71, 1991.

[21] J. Jerkert. Measurement and models of musical articulation. Master's thesis, KTH, Department of Speech Music and Hearing, 2003.

[22] B. H. Juang and L. R. Rabiner. A probabilistic distance measure for hidden markov models. *AT&T Technical Journal*, 64(2):391–408, 1985.

[23] F. Lerdahl and R. Jackendoff. *A Generative Theory of Tonal Music.* MIT Press, 1983.

[24] E. Narmour. *The Analysis and Cognition of Basic Melodic Structures.* Univ. of Chicago Press, 1990.

[25] E. Narmour. *The Analysis and Cognition of Melodic Complexity.* Univ. of Chicago Press, 1992.

[26] C. Palmer. Music performance. *Annual Review of Psychology*, 48:115–138, 1997.

[27] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *IEEE Proceedings*, 1989.

[28] C. Raphael. Automatic transcription of piano music. In D. Heckerman and C. Whittaker, editors, *Proceedings ISMIR*, Paris, France, October 2002. IRCAM.

[29] C. Raphael. A bayesian network for real-time musical accompaniment. In T.G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, number NIPS 14. MIT Press, 2002.

[30] E. Scheirer. Extracting expressive performance information from recorded music. Master's thesis, Program in Media Arts and Science, Massachusetts Institute of Technology, 1995.

[31] E. Stamatatos and G. Widmer. Automatic identification of music performers with learning ensembles. *Artificial Intelligence*, to appear, 2005.

[32] D. Temperley. *The Cognition of Basic Musical Structures.* MIT Press, 2001.

[33] A. Tobudic and G. Widmer. Playing mozart phrase by phrase. In *Proceedings of the 5th International Conference on Case-based Reasoning (ICCBR'03)*, Trondheim, Norway, 2003.

[34] A.J. Viterbi. Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Transactions of Information Theory*, IT-13:260–269, April 1967.

[35] T. Wang, N. Zheng, Y. Li, Y. Xu, and H. Shum. Learning kernel-based hmms for dynamic sequence synthesis. *Graphical Models*, 65(4):206–221, 2003.

[36] G. Widmer. Large-scale induction of expressive performance rules: First quantitative results. In *Proceedings of the International Computer Music Conference*, 2000.

[37] G. Widmer. Discovering strong principles of expressive music performance with the plcg rule learning strategy. In *Proceedings of the 12th European Conference on Machine Learning (ECML'01)*. Springer Verlag, 2001.

[38] G. Widmer and A. Tobudic. Playing mozart by analogy: Learning multi-level timing and dynamics strategies. *Journal of New Music Research*, 32(3):259–268, 2003.