# Orchestral Musical Accompaniment from Synthesized Audio

**Christopher Raphael** *
Department of Mathematics and Statistics,
University of Massachusetts at Amherst,
Amherst, MA 01003-4515,
raphael@math.umass.edu

## Abstract

We describe a computer system that synthesizes a responsive and sensitive orchestral accompaniment to a live musician in a piece of non-improvised music. The system of composed of three components "Listen," "Anticipate" and "Synthesize." Listen analyzes the soloist's acoustic signal and estimates note onset times using a hidden Markov model. Synthesize plays a prerecorded audio file back at variable rate using a phase vocoder. Anticipate creates a Bayesian network that mediates between Listen and Synthesize. The system has a learning phase, analogous to a series of rehearsals, in which model parameters for the network are estimated from training data. In performance, the system synthesizes the musical score, the training data, and the on-line analysis of the soloist's acoustic signal using a principled decision-making engine, based on the Bayesian network. A live demonstration will be given using the aria *Mi Chiamano Mimi* from Puccini's opera *La Bohème*, rendered with a full orchestral accompaniment.

## 1 Introduction

Musical accompaniment systems emulate the task that a human musical accompanist performs: supplying a missing musical part or parts, generated in real time, in response to the sound input from a live musician. As with the human musician, the accompaniment system should be flexible, responsive, able to learn from examples, and bring a sense of musicality to the task.

In 1984, Dannerberg's pioneering work [4] built from "whole cloth" an accompaniment system without the benefit of real-time digital audio input or sophisticated audio synthesis hardware. Significant improvements were developed in [5], [8], [9]; in the latter two Grubb and Dannenberg demonstrate an impressive system for the the challenging task of vocal accompaniment. Other notable efforts are Vercoe [18], Vercoe and Puckette, [19], Baird [1]. Our own work, [15], [14], [16], furthers the cause of accompaniment systems through its emphasis on probabilistic modeling and machine learning. Perhaps the most significant advantage of the machine learning approach is the ability to automatically "learn" from data examples. In particular, the use of a hidden Markov models allows an accompaniment system to learn to better "hear" the soloist. In addition, probabilistic modeling of musical performance provides a generic framework in which musical interpretation can be learned and incorporated.

Most accompaniment systems, and all cited above, create the audio accompaniment through a sparse sequence of "commands" that control a computer sound synthesis engine or dedicated audio hardware. For instance, MIDI is a natural, popular, and efficient protocol for representing and creating accompaniments. Some instruments, such as plucked string instruments, the piano, and other percussion instruments can be reasonably reproduced through the cartoon-like performance descriptions afforded by MIDI. Many other instruments, however, such as the woodwinds, brass, strings, and voice, continually vary a large number of coupled attributes during the evolution of each note or phrase. While the sparse representation of MIDI is certainly appealing for accompaniment applications, the MIDI-controllable counterparts of acoustic instruments generally sound somewhat sterile, and unconvincing.

We describe here a new direction in our work on musical accompaniment systems: we synthesize audio output by playing back an audio recording at variable rate. In doing so, the system captures a much broader range of tone color and interpretive nuance while posing a significantly

*This work supported by NSF grant IIS-0113496.

more demanding computational challenge. Our current effort intersects that of Cano, Loscos, Bonada, de Boer, and Serra [3], [10], [2] and the references therein, which builds a system for real-time morphing of vocal input for a karaoke system. Both systems perform real-time matching of audio input using HMMs; both synthesize real-time audio output that depends on the audio input; and both share a virtual-reality-like model for improving a person's musical experience. While sharing these common traits, the approaches then diverge to serve the dissimilar natures of the two applications.

Our system is composed of three components we call "Listen," "Anticipate," and "Synthesize." Listen tracks the soloist's progress through the musical score by analyzing the soloist's digitized acoustic signal. Essentially, Listen provides a running commentary on this signal, identifying times at which solo note onsets occurs, and delivering these times with variable latency. The combination of accuracy, computational efficiency, and automatic trainability provided by the hidden Markov model (HMM) framework makes HMMs well-suited to the demands of Listen. A more detailed description of the HMM approach to this problem is given in [13]. This approach shares the common HMM framework with the score-following work of Orio, Dechelle, Schwarz, [12], [11].

The actual audio synthesis is accomplished by our Synthesize module through the classic *phase vocoding* technique [7], [6]. For our purposes, the phase vocoder is an algorithm enabling variable-rate playing of an audio file without introducing pitch distortions. The Synthesize module is driven by a sequence of local synchronization goals which guide the synthesis like a trail of bread crumbs.

The sequence of local goals is the product of the Anticipate module which mediates between Listen and Synthesize. The heart of Anticipate is a Bayesian network consisting of hundreds of Gaussian random variables including both observable quantities, such as note onset times, and unobservable quantities, such as local tempo. The network can be trained during a rehearsal phase to model both the soloist's and accompanist's interpretations of a specific piece of music. This model then constitutes the backbone of a principled *real-time* decision-making engine used in live performance for scheduling musical events. A more detailed treatment of our approach to this problem is given in [15] and [16].

## 2 Listen

To follow a soloist, one must first hear the soloist; "Listen" is the component of our system that accomplishes this task.

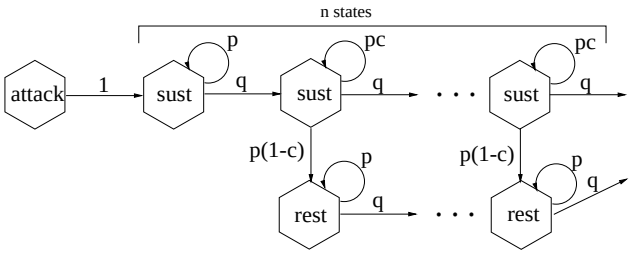


Figure 1: A Markov model for a note allowing an optional silence at the end.

We begin by dividing our acoustic signal into a collection of $N$ "snapshots" or "frames." In our current system the acoustic signal is sampled at 8 KHz with a frame size of 256 samples leading to about 31 frames per second. In analyzing the acoustic signal, we seek to label each data frame with an appropriate score position. We begin by describing the label process.

For each note in the solo part we build a small Markov model with states associated with various portions of the note such as "attack" and "sustain" as in Fig. 1. We use various graph topologies for different kinds of notes, such as short notes, long notes, rests, trills, and rearticulations. However, all of our models have tunable parameters that control the length distribution (in frames) of the note. Fig. 1 shows a model for a note that is followed by an optional silence, as would be encountered if the note is played *staccato* or if the player takes a breath or makes an expressive pause. The self-loops in the graph allow us to model of a variety of note length distributions using a small collection of states by choosing appropriate transition probabilities. We chose probabilities so that the idealized note length distributions of the model match the empirical note length distributions observed from past performances. We create a model for each solo note in the score, such as the one of Fig. 1, and chain them together in left-to-right fashion to produce the hidden label, or state, process for our HMM. We write $X_1, X_2, \ldots, X_N$ for the state process where $X_n$ is the state visited in the $n$th frame. $X_1, X_2, \ldots, X_N$ is a Markov chain.

The state process, $X_1, X_2, \ldots, X_N$, is, of course, not observable. Rather, we observe the acoustic frame data. For each frame, $n = 1, 2, \ldots$, we compute a feature vector describing the local content of the acoustic signal in that frame, $Y_n$. Most of the components of the vectors $\{Y_n\}$ are measurements derived from the finite Fourier transform of the frame data, useful for distinguishing various pitch hypotheses. Other components measure signal power, useful for distinguishing rests; and local ac-

tivity, useful for identifying rearticulations and attacks. As is consistent with the HMM model, we assume that the conditional distribution of each $Y_n$, given all other variables in our model, depends only on $X_n$.

One of the many virtues of the HMM approach is that the class conditional distributions, $p(y|x) = P(Y_n = y|X_n = x)$ can be learned in an unsupervised manner through the Baum-Welch, or Forward-Backward, algorithm. This allows our system to adapt automatically to changes in solo instrument, microphone placement, room acoustics, ambient noise, and choice of the accompaniment instrument. In addition, this automatic trainability has proven indispensable to the process of feature selection. A pair of simplifying assumptions make the learning process feasible. First, states are "tied" so that $p(y|x)$ depends only on several attributes of the state $x$ such as the associated pitch and "flavor" of state, (attack, rearticulation, sustain, etc.). The tying of the states is accomplished through the application of several simple rules, rather than learned automatically. Second, the feature vector is divided into several groups of features, $y = (y^1, \ldots, y^J)$, assumed to be conditionally independent, given the state: $p(y|x) = \prod_{j=1}^{J} p(y^j|x)$.

As important as the automatic trainability is the estimation accuracy that our HMM approach yields. Musical signals are often ambiguous locally in time but become easier to parse with the benefit of longer term hindsight. The HMM approach handles this local ambiguity naturally through its probabilistic formulation, as follows. While we are waiting to detect the $m$th solo note, we collect data (increment $n$) until

$$P(X_n \geq \text{start}_m)|Y_1 = y_1, \ldots, Y_n = y_n) > \alpha$$

for some threshold $\alpha$, where $\text{start}_m$ is the first state of the $m$th solo note model, (e.g. the "attack" state of Fig. 1). If this condition first occurs at frame $n^*$ then we estimate the onset time of the $m$th solo note by

$$\text{note}_m = \arg\max_{n \leq n^*} P(X_n = \text{start}_m|Y_1 = y_1, \ldots, Y_{n^*} = y_{n^*})$$

This latter computation is accomplished with the Forward-Backward algorithm. In this way we delay the detection of the note onset until we are reasonably sure that it is, in fact, past, greatly reducing the number of misfirings of Listen.

Finally, the HMM approach brings fast computation to our application. Dynamic programming algorithms provide the computational efficiency necessary to perform the calculations we have outlined at a rate consistent with the real-time demands of our application.

## 3 Indexing the Audio

The previous section dealt with the problem of determining note onsets for the *input* audio played by the live player. Before one can hope to resynthesize the *output* audio file to "follow" live input, one must first develop an analogous parse of the output (accompaniment) audio. We call this process *indexing* the audio. Our approach to this problem is quite similar to the HMM strategy used above. In fact, the two implementations share much of the same code. The principal differences are that indexing analysis is performed off-line and must deal with polyphonic audio. Our approach shares the basic HMM approach to score following of [12] and [11].

We begin with a score representation of the accompaniment giving the start and end times, in musical units, for each note in the score. While the score is polyphonic, we reduce the score to a one-dimensional representation — a string of "chords" — as follows. At each point in the musical score a number of possibly repeated pitches collectively sounds in the ensemble. Whenever this collection of pitches changes, we add a new chord to our sequence. This sequence of chords is essentially a homophonic representation of polyphonic music that disregards the partitioning of the notes into voices. Each chord in the sequence has an associated musical duration, so our model understands the composite rhythm of the score.

We build a Markov chain for the traversal of the score in an identical manner to that of Section 2. This model serves as the hidden process of our HMM. As in Section 2 the Markov model encodes our prior assessment of the different chord length distributions, (without the benefit of past performances).

The *data model* for our indexing problem describes the probability of a single frame of data given a known chord configuration. The modeling problem is complicated by the polyphonic nature of our data. Due to the highly variable nature of orchestral texture and timbre, we have chosen to build a model by hand, rather than learning a model from real data that is unlikely to generalize. This approach is in contrast to the fully learned data model of Section 2 that deals with monophonic known-instrument data.

An example of a section of our data can be seen in Fig. 2, taken from the orchestra part of the aria *Mi Chiamano Mimi* from Puccini's opera *La Bohème*. In this spectrogram representation we divide the 8KHz (down-sampled from 48KHz) audio data into a sequence of overlapping frames of 1024 samples each. Our goal is to find the start time, in frames, of each chord.

Figure 2: A spectrogram of an excerpt from the orchestral accompaniment to Puccini's *Mi Chiamano Mimi*.

First we suppose a single note is sounding at frequency $c$. In this case we would expect to see spectral energy distributed primarily at frequencies in the overtone series of the note: $c, 2c, 3c, \ldots$. Thus we build a probability density on frequency by

$$g(\omega; c) = \sum_{m=1}^{M} p_m N(\omega; mc, \sigma^2)$$

where $\sum_{m=1}^{M} p_m = 1$ and $N(\omega; \mu, \sigma^2)$ is the normal density function. The number of harmonics considered, $M$, and the way in which they decay $p_1, \ldots, p_M$ are parameters of our model. Suppose we are given a spectrum $s = (s_1, \ldots, s_F)$ where $s_f$ is the energy (squared modulus) of the $f$th frequency bin, and imagine that the units are such that nothing significant is lost by treating the components of $s$ as integers. Thus our spectrum has $s_1$ energy units at frequency bin 1, $s_2$ energy units at frequency bin 2, etc., We view these energy units as independent samples from a discretized version $g = (g_1, \ldots, g_F)$ of $g(\omega; c)$. Thus the probability of the spectrum would be

$$p(s|c)^1 = \prod_{f=1}^{F} g_f^{s_f} \tag{1}$$

There is really no need to require that $s$ is composed of integral values or restrict our attention to single notes. To this end let $c = (c_1, \ldots, c_H)$ be a collection of fundamental frequencies and define

$$g(\omega; c) = \sum_{h=1}^{H} \sum_{m=1}^{M} p_m N(\omega; mc_h, \sigma^2)$$

and let

$$p(s|c) = \prod_{f=1}^{F} g_f^{s_f/\alpha}$$

where $(g_1, \ldots, g_F)$ is again the discretized version of $g(\omega; c)$.

[1] More precisely, this is $P(s|c, t)$ where $t = \sum_f s_f$ is the total amount of energy in the spectrum. Since the HMM depends only on that way $p(s|c)$ varies as a function of $c$, nothing is lost by ignoring this detail.

The parameters of our model, $\sigma, M, \alpha, p_1, \ldots p_M$ should ideally be learned from training data. In fact, we anticipate addressing more complex models with many more unknown parameters in the future, virtually necessitating automatic learning of parameters. For our example here, however, the parameters were set by hand with reasonable results.

We parsed the data by choosing the onset for the $m$th chord as

$$\text{onset}_m = \arg\max_n P(X_n = \text{start}_m | Y = y)$$

where $X = (X_1, \ldots, X_N)$ is our hidden process, $Y = y$ is our spectrogram, and $\text{start}_m$ is the first state of the $m$th chord. This computation is performed with the forward-backward algorithm. An example of the parse obtained by this method can be heard at http://fafner.math.umass.edu/icmc03. While this computation saved considerable effort, we still edited the results by hand to ensure greater accuracy.

## 4 Synthesize

The Synthesize module takes as input both an audio recording of the accompaniment and the index into the recording. The role of Synthesize is to play this recording back at variable rate (and without pitch change) so that the accompaniment *follows* the soloist. The variable playback rate is accomplished using a phase vocoder [7], [6]. This technique begins by dividing the signal into a sequence of overlapping windows indexed by $k$ and computing the short time Fourier transform, $F_k$, for each window. Both the magnitude, $|F_k|$, and the *phase difference* between consecutive windows, $\Delta_k = \arg(F_k) - \arg(F_{k-1})$ — both functions of frequency — are saved. The $n$th frame of output is constructed using an *accumulated phase* function $\phi_n$. We initialize $\phi_0$ arbitrarily and compute the $n$th output frame by choosing a window, $k(n)$ and computing the inverse Fourier transform of the complex function with $|F_{k(n)}|$ as magnitude and $\phi_n$ as phase. The accumulated phase is then incremented by $\phi_{n+1} = \phi_n + \Delta_{k(n)}$. The phase vocoder ensures that the phase changes in a smooth manner from frame to frame.

As an example, the audio data could be played at double the rate by letting $k(n+1) = k(n) + 2/\Omega$ for $n = 1, 2, \ldots$ where $\Omega$ is the fraction of a window that does not overlap with its successor. We refer to this rate of progress, 2 in this example, as the "play rate."

The literature on musical synthesis is rich; we regard the phase vocoder as a "place-holder" in a larger effort, that can be improved upon at a later time.

In our application the play rate varies with time and must be chosen so that the solo and accompaniment synchronize while maintaining a reasonably smooth playback of the audio data. Subsequent sections focus on creating a sequence of short term "goals" for Synthesize — times at which the next unplayed accompaniment note should sound. When a new goal is set, Synthesize calculates the play rate necessary to achieve the goal and follows the new play rate until the play rate is reset.

In our experiments we used an output sampling rate of 48 KHz with a frame size of 4096 samples and an overlap rate of $\Omega = 1/4$. Output frames are smoothed using an "overlap-add" technique.

# 5 Anticipate

A musical accompaniment requires the synthesis of a number of different knowledge sources. From a modeling perspective, the fundamental challenge of musical accompaniment is to express these disparate knowledge sources in terms of a common denominator. We describe here the three knowledge sources we use.

We work with non-improvisatory music so naturally the musical score, which gives the pitches and relative durations of the various notes, as well as points of synchronization between the soloist and accompaniment, must figure prominently in our model. The score should not be viewed as a rigid grid prescribing the precise times at which musical events will occur; rather, the score gives the basic elastic material which will be stretched in various ways to produce the actual performance. The score simply does not address most interpretive aspects of performance.

Since our accompanist must follow the soloist, the output of the Listen component, which identifies note boundaries in the solo part, constitutes our second knowledge source. Given the variable latency in the communication of detections from Listen, we feel that any successful accompaniment system cannot synchronize in a purely responsive manner. Rather it must be able to predict the future using the past and base its synchronization on these predictions, as human musicians do.

While the same player's performance of a particular piece will vary from rendition to rendition, many aspects of musical interpretation are clearly established with only a few examples. These examples constitute the third knowledge source for our system. The solo data, (solo note onset times estimated from past rehearsals), are used primarily to teach the system how to predict the future evolution of the solo part. The accompaniment data, (the accompaniment onset times estimated from the accompaniment recording), are used to bias the system toward the interpretation exhibited in the recording as well as toward a uniform play rate.

We have developed a probabilistic model, a Bayesian network, that represents all of these knowledge sources through a jointly Gaussian distribution containing potentially thousands of random variables. The observable variables in this model are the estimated soloist note onset times produced by Listen and the onset times for the accompaniment notes. Between these two layers of observable variables lies a layer of hidden variables that describe unobservable (and imaginary) quantities such as local tempo, change in tempo, and rhythmic stress.

## 5.1 A Model for Rhythmic Interpretation

We begin by describing a model for the sequence of note onset times generated by a monophonic (single voice) musical instrument playing a known rhythm. For each of the notes, indexed by $n = 0, \ldots, N$, we define a random vector representing the time, $t_n$, (in seconds) at which the note begins, and the local "tempo," $s_n$, (in secs. per measure) for the note. We model this sequence of random vectors through a random difference equation:

$$\begin{pmatrix} t_{n+1} \\ s_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & l_n \\ 0 & 1 \end{pmatrix} \begin{pmatrix} t_n \\ s_n \end{pmatrix} + \begin{pmatrix} \tau_n \\ \sigma_n \end{pmatrix} \quad (2)$$

$n = 0, \ldots, N-1$, where $l_n$ is the musical length of the $n^{\text{th}}$ note, in measures, and the $\{(\tau_n, \sigma_n)^t\}$ and $(t_0, s_0)^t$ are mutually independent Gaussian random vectors.

The distributions of the $\{\sigma_n\}$ will tend concentrate around 0 expressing the notion that tempo changes are gradual. The means and variances of the $\{\sigma_n\}$ show where the soloist is speeding-up (negative mean), slowing-down (positive mean), and tell us if these tempo changes are nearly deterministic (low variance), or quite variable (high variance). The $\{\tau_n\}$ variables also concentrate around 0 and describe stretches (positive mean) or compressions (negative mean) in the music that occur without any actual change in tempo, as in a *tenuto* or *agogic* accent. The addition of the $\{\tau_n\}$ variables leads to a more musically plausible model, since not all variation in note lengths can be explained through tempo variation. Equally important, however, the $\{\tau_n\}$ variables stabilize the model by not forcing the model to explain, and hence respond to, all note length variation as tempo variation.

Collectively, the distributions of the $(\tau_n, \sigma_n)^t$ vectors characterize the rhythmic interpretation. Both overall tendencies (means) and the repeatability of these tendencies (covariances) are captured by these distributions.

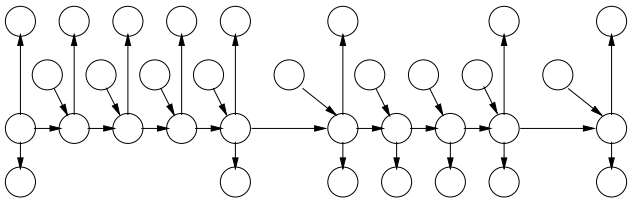

Figure 3: A graphical description of the dependency structure of our model.

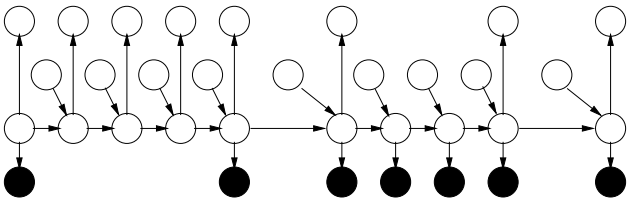

Figure 4: Conditioning on the observed accompaniment performance (darkened circles), we use the message passing algorithm to compute the conditional distributions on the unobservable $\{\tau_n, \sigma_n\}$ variables.

#### 5.1.1 Joint Model of Solo and Accompaniment

In modeling the situation faced by our accompaniment system, we begin with the basic rhythm model of Eqn. 2, now applied to the *composite rhythm*. More precisely, let $m_0^s, \ldots, m_{N^s}^s$ and $m_0^a, \ldots, m_{N^a}^a$ denote the positions, in measures, of the various solo note onsets and accompaniment events, where by the latter we mean an onset or termination of an accompaniment note in any voice. We then let $m_0, \ldots, m_N$ be the sorted union of these two sets of positions with duplicate times removed; thus $m_0 < m_1 < \ldots < m_N$. We then use the model of Eqn. 2 with $l_n = m_{n+1} - m_n$, $n = 0, \ldots, N-1$. A graphical description of this model is given in the middle two layers of Figure 3. In this figure, the 3rd layer from the top corresponds to the time-tempo variables, $(t_n, s_n)^t$, for the composite rhythm, while the 2nd layer from the top corresponds to the *interpretation* variables $(\tau_n, \sigma_n)^t$. The directed arrows of this graph indicate the conditional dependency structure of our model. Thus, given all variables "upstream" of a variable in the graph, the conditional distribution of that variable depends only on its parent variables.

Recall that the Listen component estimates the times at which solo notes begin. How do these estimates figure into our model? We model the note onset times estimated by Listen as noisy observations of the true positions $\{t_n\}$. Thus, if $m_n$ is a measure position at which a solo note occurs, then the corresponding estimate from Listen is modeled as

$$a_n = t_n + \alpha_n$$

where $\alpha_n \sim N(0, \nu^2)$. Similarly, if $m_n$ is the measure position of an accompaniment event, then we model the observed time at which the event occurs as

$$b_n = t_n + \beta_n$$

where $\beta_n \sim N(0, \eta^2)$. The note onsets estimated by Listen constitute the top layer of our figure while the accompaniment event times constitute the bottom layer. There are, of course, measure positions at which both solo and accompaniment events should occur. If $n$ indexes such a time then $a_n$ and $b_n$ will both be noisy observations of the true time $t_n$.

The vectors/variables $(t_0, s_0)^t$ and $\{(\tau_n, \sigma_n)^t, \alpha_n, \beta_n\}_{n=1}^N$ are assumed to be mutually independent.

### 5.2 The Rehearsal Phase

Our system learns its rhythmic interpretation by estimating the parameters of the trainable ($(t_0, s_0)^t$ and $\{(\tau_n, \sigma_n)^t\}_{n=1}^N$) variables through a procedure analogous to a series of rehearsals. We initialize the model parameters — the means and covariances of the trainable variables — to a "neutral" interpretation, and perform with our system as described in Section 5.3. Each such rehearsal results in an audio file which we parse in an off-line fashion, as in Section 2 to produce a sequence of times at which solo note onsets occurred. These sequences of observed times, along with the index into the accompaniment, serve as our training data.

We treat each sequence of times as a collection of observed variables in our belief network. For instance, the accompaniment times are shown with darkened circles in Figure 4. Given an initial assignment of means and covariances to the trainable variables, we use the "message passing" algorithm of Bayesian networks [17], to compute the conditional distributions (given the observed performance) of the trainable variables. While we don't observe the trainable variable directly, we can think of the means of these conditional distributions as observations of the variables. Several performances then lead to several observations of each variable and natural estimation algorithms follow. For instance, we could take the empirical mean and covariance of the conditional means as our parameter estimates. The details of the learning algorithm are presented in [15].

### 5.3 Real Time Accompaniment

The methodological key to our real-time accompaniment algorithm is the computation of (conditional) marginal

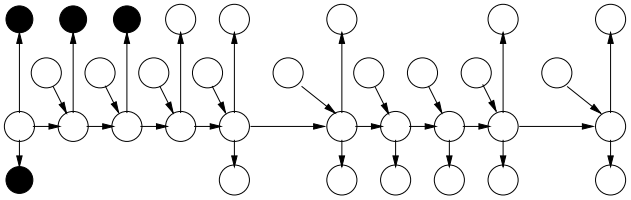

Figure 5: At any given point in the performance we will have observed a collection of solo note times estimated estimated by Listen, and the accompaniment event times (the darkened circles). We compute the conditional distribution on the next unplayed accompaniment event, given these observations.

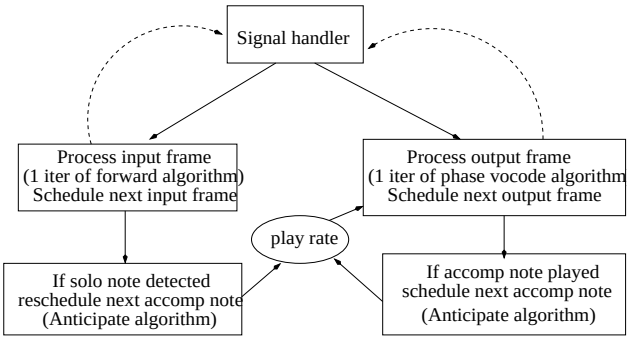


Figure 6: The flow of execuation of the real-time part of our program is completely driven by signals, as described in the figure.

distributions facilitated by the message-passing machinery of Bayesian networks. At any point during the performance some collection of solo notes and accompaniment events will have been observed, as in Fig. 5. Conditioned on this information we can compute the distribution on the next unplayed accompaniment event. The real-time computational requirement is limited by passing only the messages necessary to compute the marginal distribution on the pending accompaniment event.

Once the conditional marginal distribution of the pending accompaniment event is calculated, we schedule the event accordingly (reset the play rate). Currently we schedule the event to be played at the conditional mean time, given all observed information, however other reasonable choices are possible. Note that this conditional distribution depends on *all* of the sources of information included in our model: The score information, all currently observed solo and accompaniment event times, and the rhythmic interpretations demonstrated by both the soloist and accompanist, learned during the training phase.

The initial scheduling of each accompaniment event takes place immediately after the previous accompaniment event is played. It is possible that a solo note will be detected before the pending accompaniment event is played; in this case, the pending accompaniment event is rescheduled by recomputing its conditional distribution using the newly available information and resetting the play rate. This happens every time a new solo note is detected,until the event is finally played.

## 6 Computation

Our program must manage three types of computations and these are organized through two separate asynchronous callback "loops." First, the Listen module analyzes our acoustic input at a rate of about 31 frames per second. Thus the basic iteration of Listen processes an input frame and sets a signal instructing the Listen module to "wake up" when the next input frame is ready. The Synthesize module works analogously processing output frames at a rate of about 47 frames per second, also driven by signal callback. Due to the time-critical nature of our Synthesize module, Synthesize callbacks always take precedence over Listen callbacks.

When an input frame is processed, we read the next frame of sound data from the audio device and perform one iteration of the HMM analysis engine. If this iteration results in the detection of a solo note, we fix the appropriate variable of our belief network to the estimated solo note onset time and recompute the time of the pending accompaniment event. This results in a new target time for the pending accompaniment event thus changing the play rate for subsequent iterations of Synthesize.

Each iteration of Synthesize computes and plays a new frame of output data using our phase vocoder with the current play rate. When the Synthesize module plays a frame crossing an accompaniment event boundary (as represented by the index), we consider that an accompaniment event has been played. We then fix the corresponding variable of our belief network and perform our initial scheduling of the next accompaniment event. We choose the new play rate so that the next accompaniment event will be played at the time recommended by this computation.

Once audio data is written we cannot easily revise these audio samples. For this reason we wish to minimize the time between each write command and that when the corresponding samples "cross the speaker." However, when this time becomes too short, inaccuracy in the delivery of signal callbacks leads to audio underruns. Our current system writes samples .025 seconds before they are actually played.

Our program is written in c and runs on a 1.6 GHz Pentium 4 Linux notebook using the ALSA audio driver.

## 7 Live Demonstration

Our conference presentation will feature a live demonstration of the system on the aria "Mi Chiamano Mimi" from Puccini's opera La Bohème performed by the author. The accompaniment audio file for this aria was taken from a MusicMinusOne$^{TM}$ recording. An audio example of such a performance can be heard at http://fafner.math.umass.edu/accompaniment/mimi.html along with a number of other audio examples of related accompaniment efforts.

## References


[1] B. Baird, D. Blevins, and N. Zahler. Artificial intelligence and music: Implementing an interactive computer performer. *Computer Music Journal*, 17(2):73–79, 1993.

[2] Pedro Cano, Alex Loscos, Jordi Bonada, Marten de Boer, and Xavier Serra. Singing voice impersonator application for pc. In *Proc. Int. Comp. Music Conf.* Int. Computer Music Assoc., 2000.

[3] Pedro Cano, Alex Loscos, Jordi Bonada, Marten de Boer, and Xavier Serra. Voice morphing system for impersonating in karaoke applications. In *Proc. Int. Comp. Music Conf.* Int. Computer Music Assoc., 2000.

[4] R. Dannenberg. An on-line algorithm for real-time accompaniment. In *Proceedings of the International Computer Music Conference, 1984*, pages 193–198. Int. Computer Music Assoc., 1984.

[5] R. Dannenberg and H. Mukaino. New techniques for enhanced quality of computer accompaniment. In *Proceedings of the International Computer Music Conference, 1988*, pages 243–249. Int. Computer Music Assoc., 1988.

[6] M. Dolson. The phase vocoder: A tutorial. *Computer Music Journal*, 10(4):14–27, 1986.

[7] J. L. Flanagan and R. M. Golden. Phase vocoder. *Bell System Technical Journal*, pages 1493–1509, Nov. 1966.

[8] Lorin Grubb and Roger Dannenberg. A stochastic method of tracking a vocal performer. In *Proc. Int. Comp. Music Conf.*, pages 301–308. Int. Computer Music Assoc., 1997.

[9] Lorin Grubb and Roger Dannenberg. Enhanced vocal performance tracking using multiple information sources. In *Proc. Int. Comp. Music Conf.*, pages 37–44. Int. Computer Music Assoc., 1998.

[10] Alex Loscos, Pedro Cano, and Jordi Bonada. Low-delay singing voice alignment to text. In *Proc. Int. Comp. Music Conf.* Int. Computer Music Assoc., 1999.

[11] N. Orio and Schwarz D. Alignment of monophonic and polyphonic music to a score. In *Proc. Int. Comp. Music Conf.*, pages 155–158. Int. Computer Music Assoc., 2001.

[12] N. Orio and F. Dechelle. Score following using spectral analysis and hidden markov models. In *Proc. Int. Comp. Music Conf.*, pages 151–154. Int. Computer Music Assoc., 2001.

[13] C. Raphael. Automatic segmentation of acoustic musical signals using hidden markov models. *IEEE Trans. on PAMI*, 21(4):360–370, 1999.

[14] C. Raphael. Music plus one: A system for expressive and flexible musical accompaniment. In *Proc. Int. Comp. Music Conf.* Int. Computer Music Assoc., 2001.

[15] C. Raphael. A probabilistic expert system for automatic musical accompaniment. *Jour. of Comp. and Graph. Stats.*, 10(3):487–512, 2001.

[16] 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, NIPS 14*. MIT Press, 2002.

[17] D. Spiegelhalter, A. P. Dawid, S. Lauritzen, and R. Cowell. Bayesian analysis in expert systems. *Statistical Science*, 8(3):219–283, 1993.

[18] B. Vercoe. The synthetic performer in the context of live performance. In *Proceedings of the International Computer Music Conference, 1984*, pages 199–200. Int. Computer Music Assoc., 1984.

[19] B. Vercoe and M. Puckette. Synthetic rehearsal: Training the synthetic performer. In *Proceedings of the International Computer Music Conference, 1985*, pages 275–278. Int. Computer Music Assoc., 1985.