Orchestra in a Box: A System for Real-Time Musical Accompaniment

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 plays a responsive and sensitive 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*.

1 Introduction

Musical accompaniment systems seek to emulate the task that a human musical accompanist performs: supplying a missing musical part, 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.

Most, if not all, efforts in this area create the audio accompaniment through a sparse sequence of "commands" that control a computer sound synthesis engine or dedicated audio hardware [Dannenberg, 1984], [R. Dannenberg, 1988], [B. Vercoe, 1985], [B. Baird, 1993]. For instance MIDI (musical instrument digital interface), the most common control protocol, generally describes each note in terms of a start time, an end time, and a "velocity." Some instruments, such as plucked string instruments, the piano, and other percussion instruments can be reasonably reproduced through such cartoon-like performance descriptions. Most instruments, however, continually vary several attributes during the evolution of each note or phrase, thus their MIDI counterparts sound reductive 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 more demanding computational challenge.

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 [Raphael, 1999].

The actual audio synthesis is accomplished by our Synthesize module through the classic *phase vocoding* technique. Essentially, 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 various approaches to this problem is given in [Raphael, 2001] and [Raphael, 2002].

[*]This work supported by NSF grants IIS-998789 and IIS-0113496.

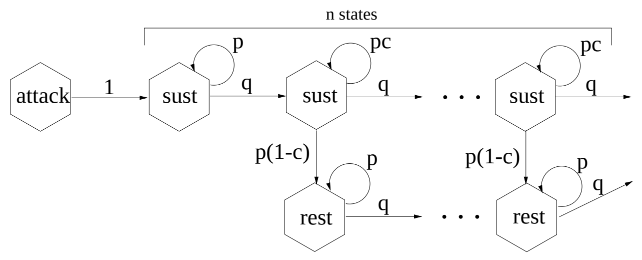


Figure 1: A Markov model for a note allowing an optional silence at the end.

2 Listen

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

We begin by dividing our acoustic signal into a collection of "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.

The "score" containing both solo and accompaniment parts is known to us. Using this information, 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. For instance, one can show that, for the model of Fig. 1, the number of frames following that attack state has a Negative Binomial distribution, $NB(n,p)$, where n and p are as described in the figure. 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 $X_1, X_2, \ldots$ for the state process where X_k is the state visited in the kth frame. $X_1, X_2, \ldots$ is a Markov chain.

The state process, $X_1, X_2, \ldots$, is, of course, not observable. Rather, we observe the acoustic frame data. For each frame, $k = 1, 2, \ldots$, we compute a feature vector describing the local content of the acoustic signal in that frame, Y_k. Most of the components of the vectors $\{Y_k\}$ 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 activity, useful for identifying rearticulations and attacks. As is consistent with the HMM model, we assume that the conditional distribution of each Y_k, given all other variables in our model, depends only on X_k.

One of the many virtues of the HMM approach is that the class conditional distributions, $p(y|x) = P(Y_k = y | X_k = x)$ 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.). 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 mth solo note, we collect data (increment k) until

$$P(X_k \geq \text{start}_m) | Y_1 = y_1, \ldots, Y_k = y_k) > \alpha$$

for some threshold α, where start_m is the first state of the mth solo note model, (e.g. the "attack" state of Fig. 1). If this condition first occurs at frame k^* then we estimate the onset time of the mth solo note by

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

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 Synthesize

The Synthesize module takes as input both an audio recording of the accompaniment as well as an "index" into the recording consisting of times at which the various accompaniment notes are played. The index is calculated through an offline variant of Listen applied to the accompaniment audio data. The polyphonic and multitimbral nature of the audio data make for a difficult estimation problem. To ensure that our index is accurate we hand-corrected the results after the fact in the experiments we will present.

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 [Flanagan and Golden, 1966], [Dolson, 1986]. 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 nth frame of output is constructed using an *accumulated phase* function ϕ_n. We initialize ϕ_0 arbitrarily and compute the nth 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 ϕ_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 Ω 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." Many improvements of this basic technique are possible which might lead to more realistic audio output.

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.

4 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 thousands of random variables. The observable variables in this model are the estimated soloist note onset times produced by Listen and the onset observable times for the accompaniment notes. Between these two layers of observable variables lies a layer of hidden variables that describe unobservable quantities such as local tempo, change in tempo, and rhythmic stress.

4.1 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 piece of music. 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:

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

$n = 0, \ldots, N-1$, where l_n is the musical length of the n^{th} note, in measures, and the $\{(\tau_n, \sigma_n)^t\}$ and $(t_0, s_0)^t$ are mutually independent Gaussian random vectors. For instance, for a half note in 4/4 time $l_n = 1/2$, whereas a half note in 3/4 time would have $l_n = 2/3$.

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 solo player's rhythmic interpretation. Both overall tendencies (means) and the repeatability of these tendencies (covariances) are captured by these distributions.

Joint Model of Solo and Accompaniment

In modeling the situation of musical accompaniment we begin with our basic rhythm model of Eqn. 1, now applied to the *composite rhythm*. More precisely, let $m_0^s, \ldots, m_{N^s}^s$ and

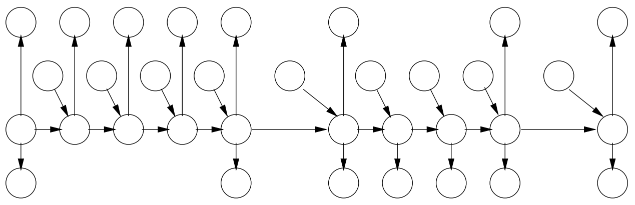


Figure 2: A graphical description of the dependency structure of our model. The top layer of the graph corresponds to the solo note onset times detected by Listen. The 2nd layer of the graph describes the (τ_n, σ_n) variables that characterize the rhythmic interpretation. The 3rd layer of the graph is the time-tempo process $\{(s_n, t_n)\}$. The bottom layer is the observed accompaniment event times.

$m_0^a, \ldots, m_{N^a}^a$ 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 any accompaniment note. 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. 1 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 2. 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.

4.2 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

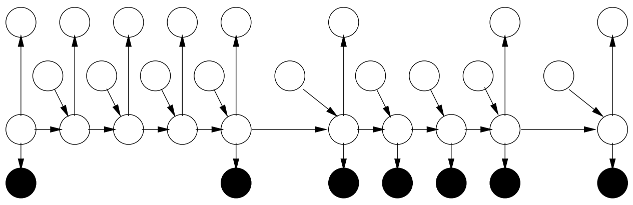


Figure 3: Conditioning on the observed accompaniment performance (darkened circles), we use the message passing algorithm to compute the conditional distributions on the unobservable $\{\tau_n, \sigma_n\}$ variables.

and covariances of the trainable variables — and perform with our system as described in Section 4.3. Each such rehearsal results in an audio file which we parse in an off-line fashion to produce a sequence of times at which solo note onsets occurred. These sequences of observed times, along with the sequence of accompaniment event times estimated from the original audio recording, 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 3. Given an initial assignment of means and covariances to the trainable variables, we use the "message passing" algorithm of Bayesian networks [Spiegelhalter *et al.*, 1993], [Cowell *et al.*, 1999], to compute the conditional distributions (given the observed performance) of the trainable variables.

We then perform analogous computations with the solo performances leading to several sets of conditional distributions for the trainable variables. These are used to reestimate the parameters of the trainable distributions using the EM algorithm.

More specifically, we estimate the trainable parameters as follows, and as in [Lauritzen, 1995]. We let μ_n^0, Σ_n^0 be our initial mean and covariance matrix for the vector (τ_n, σ_n). We assume we have J sequences of observed variables, one corresponding to the accompaniment performance and the remaining ones taken from solo performances. During the ith iteration of the algorithm, the conditional distribution of (τ_n, σ_n) given the jth sequence, and using $\{\Sigma_n^i, \mu_n^i\}$, has a $N(m_{j,n}^i, S_n^i)$ distribution where the $m_{j,n}^i$ and S_n^i parameters are computed using the message passing algorithm. We then update our parameter estimates by

$$\begin{aligned} \mu_n^{i+1} &= \frac{1}{J} \sum_{j=1}^{J} m_{j,n}^i \\ \Sigma_n^{i+1} &= S_n^i + \frac{1}{J} \sum_{j=1}^{J} (m_{j,n}^i - \mu_n^{i+1})(m_{j,n}^i - \mu_n^{i+1})^t \end{aligned}$$

An identical computation is performed to estimate the mean and covariance of (s_0, t_0).

4.3 Real Time Accompaniment

The methodological key to our real-time accompaniment algorithm is the computation of (conditional) marginal distributions facilitated by the message-passing machinery of

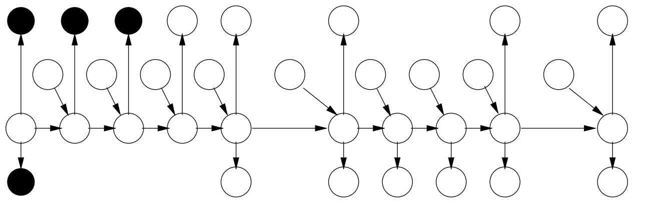


Figure 4: At any given point in the performance we will have observed a collection of solo note times estimated estimated by Listen, and the accompaniment event times (the darkened circles). We compute the conditional distribution on the next unplayed accompaniment event, given these observations.

Bayesian networks. At any point during the performance some collection of solo notes and accompaniment events will have been observed, as in Fig. 4. 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. 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.

A rather interesting case can be made for scheduling notes slightly later than the conditional mean. If an accompaniment event corresponding to a point of synchronicity with the soloist is scheduled late, then when the solo note is detected the accompaniment event will be rescheduled (and played immediately). On the other hand, an accompaniment event that is scheduled early is simply played early. Thus, the consequences of scheduling an accompaniment event at a particular time are not symmetric around the actual solo note time.

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. The pending accompaniment event is rescheduled each time an additional solo note is detected, until its currently scheduled time arrives, at which time it is finally played. In this way, every time an accompaniment note is played, it's time depends on all currently available information.

In principle, each desired marginal distribution could be computed by performing "Collect Evidence" with the clique containing the desired variable as root. That is, passing messages inward toward the root, while observing the constraint that each clique sends a message only after it has received all incoming messages. Given the rather unusual situation

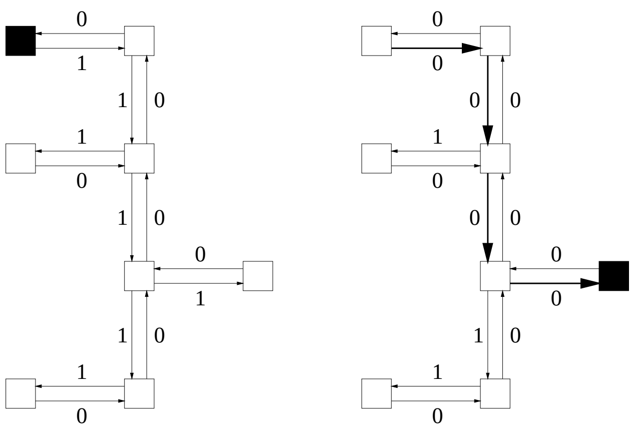


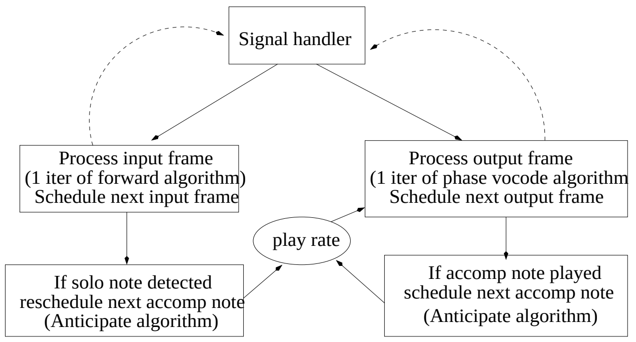
Figure 5: Abbreviated version of "Collect Evidence"

we face, in which our algorithm both observes variables and computes marginals at interspersed, and *a priori* unknown, times, a full computation of "Collect Evidence" is needlessly wasteful. Instead we have implemented the following approach.

Off-line we perform a complete round of the message passing algorithm, thereby leaving our junction tree in equilibrium. Thus, each clique contains the marginal distribution for the associated collection of variables. Each pair of neighboring cliques in the junction tree are connected by two *directed* edges, and we initially mark each of these edges as "0," meaning that no change in the probability representation will result of a message is passed along the edge. This is, of course, true since the junction tree is in equilibrium. Each time a variable is observed, we first identify a clique containing that variable. Then every directed edge moving away from that clique in the junction tree is marked as "1," as in the left panel of Fig. 5. There are pending messages that wait to be passed along these edges. When we wish to compute a marginal distribution, we identify a clique containing the variable of interest — the darkened square in the right panel of Figure 5. We then perform an abbreviated version of "Collect Evidence" using that clique as root. However, in passing messages toward the root, we only pass those marked as "1," since the other messages will not affect our probability representation. After each message is passed over a directed edge marked "1," that edge is then reset to "0," as in the right panel of Fig. 5. We are justified in marking these recently traversed edges as "0" since any other message sent along such an edge will not affect the probability representation. This reduces the necessary computation considerably.

5 Computation

Our program must manage three types of computations and these are organized through two separate asynchronous callback "loops," as depicted in Figure 6. 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



Signal handler
Process input frame
(1 iter of forward algorithm)
Schedule next input frame
Process output frame
(1 iter of phase vocode algorithm
Schedule next output frame
play rate
If solo note detected
reschedule next accomp note
(Anticipate algorithm)
If accomp note played
schedule next accomp note
(Anticipate algorithm)

Figure 6: Signals are delivered to the signal handler — the top box in the figure. This will cause either an input frame to be processed (the left branch), or an output frame to be processed and played (right branch). Either case will result in the scheduling of a new signal.

about 47 frames per second, also driven by signal callback.

When an input frame is processed, we read the next frame of sound data 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 through "Collect Evidence." This results in a new target time for the pending accompaniment note 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 note boundary, we consider that an accompaniment note has been played. We then fix the corresponding variable of our belief network and perform our initial scheduling of the next accompaniment note using "Collect Evidence." We choose the new play rate so that the next accompaniment note will be played at the time recommended by this computation.

Our program is written in c and runs on a 1.6 GHz Pentium 4 Linux notebook.

6 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$^{\mathrm{TM}}$ recording. An example of our current state of the art with this project can be heard at http://fafner.math.umass.edu/ijcai03 along with a number of other audio examples of related accompaniment efforts.

References

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

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

[Cowell *et al.*, 1999] R. Cowell, A. P. Dawid, S. Lauritzen, and D. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer, New York, New York, 1999.

[Dannenberg, 1984] 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.

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

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

[Lauritzen, 1995] S. L. Lauritzen. The em algorithm for graphical association models with missing data. *Computational Statistics and Data Analysis*, 19:191–201, 1995.

[R. Dannenberg, 1988] H. Mukaino R. Dannenberg. 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.

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

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

[Raphael, 2002] 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.

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