Query By Humming: Finding Songs in a Polyphonic Database

John Duchi
Computer Science Department
Stanford University
jduchi@stanford.edu

Benjamin Phipps
Computer Science Department
Stanford University
bphipps@stanford.edu

December 16, 2005

Abstract

Query by humming is the problem of retrieving musical performances from hummed or sung melodies. This task is complicated by a wealth of factors, including noisiness of input signals from a person humming or singing, variations in tempo between recordings of pieces and queries, and accompaniment noise in the pieces we are seeking to match. Previous studies have most often focused on the problems of retrieving melodies represented symbolically (as in MIDI format), in monophonic (single voice or instrument) audio recordings, or retrieving audio recordings from correct MIDI or other symbolic input melodies. We take a step toward developing a framework for query by humming in which polyphonic audio recordings can be retrieved by a user singing into a microphone.

1 Introduction

Suppose we hear a song on the radio but either do not catch its title or simply cannot remember it. We find ourselves with songs stuck in our heads and no way to find the songs save visiting a music store and singing to the music clerk, who can then (hopefully) direct us to the pieces we want. Automating this process seems a reasonable goal.

The first task in such a system is to retrieve pitch from a human humming or singing. There is a large literature on retrieving pitches from voice via a machine. There are many algorithms to detect pitch; most rely on a combination of different calculations. Often, a sliding window of 5 to 10 ms intervals is preprocessed to gain initial estimates of pitch, then windowed autocorrelation functions [4] or a power spectrum analysis is done. After these steps, there is often interpolation and postprocessing on the sound data to remove errors such as octave off problems [1] to give a series of frequencies and the times at which the frequencies are estimated.

The second task in a query by humming system is to take the pitches and the durations that have been calculated to find the actual recording represented. There has been research in this area before, but most has been using music stored in MIDI or some other symbolic formats [5] or in monophonic (single voice) recordings [6]. In real polyphonic recordings, a number of factors complicate queries–these include high tempo variability, which depends on specific performances, and the inconsistencies of the spectrum of sound due to factors such as instrument timbre and vibrato.

To move beyond the above listed difficulties of making queries over polyphonic recordings, we basae our algorithms on a generative probabilistic model developed by Shalev-Shwartz et al. [2]. This builds on work in dynamic Bayesian networks and HMMs [5] to create a joint probability model over both temporal and spectral probabilistic components of our polyphonic recordings to give us a retrieval procedure for our sung queries.

2 Problem Representation

Though there are two parts to our problem–pitch extraction and retrieving musical performances given a melody–the latter necessitates the most detailed problem setting. Formally (using notation essentially identical to that of [2]), we are able to define the set of possible pitches Γ (in Hz), in well-tempered Western music tuning, as $\Gamma = \{440 \cdot 2^{s/12} | s \in \mathbb{Z}\}$. Thus, a melody is a sequence of pitches $\mathbf{p} \in \Gamma^k$, where k is the length of the melody (in notes).

For our purposes, the real performance of a melody is a discrete time sampled audio signal, $\mathbf{o} = o_1, \ldots, o_T$, where o_t is the spectrum of one of our performances at the t^{th} discrete sample. These performances are those drawn from our database of pieces that we query. Because we assume short-time invariance of our input sounds, we set the samples to be of length .04 seconds.

To completely define a melody, we have a series of k pitches p_i and durations d_i, where the melody is to play p_1 for d_1 seconds and so on. Performances of pieces, however, rarely use the same tempo, and thus a melody can have much more variability than the model given. As such, we define a sequence of scaling factors for the tempo of our queries, $\mathbf{m} \in (\mathbb{R}^+)^k$, the set of sequences of k positive real numbers (in our testing, each m_i is drawn from a set M of all the possible scaling factors). Thus, the actual duration of p_i is $d_i m_i$, which we must take into account when matching queries to audio signals.

Now we have our problem defined: given a melody $\langle \mathbf{p}, \mathbf{d} \rangle$, we would like to find the likelihood of some performance, that is, we would like to maximize $\mathbf{o}$ in our generative model $P(\mathbf{o}|\mathbf{p}, \mathbf{d})$.

3 Extracting Pitch

Having defined our problem, we see that the first step must be to extract pitches and durations from a sung query. Saul et. al., have described an algorithm that does not rely on power spectrum analysis or long autocorrelations to find pitches in voice [1]. Their algorithm (which is called Adaptive Least Squares - ALS) uses least squares approximations to find the optimal frequency values of a signal. A method known as Prony's method [3] uses only one-sample lagged and zero-sample lagged autocorrelation as well as least squares, which reduces errors in resolution sometimes found by FFTs as a result of low sampling rates, and we can extract pitches in time linear in the number of samples we have.

3.1 Finding the Sinusoid in Voiced Speech

Any sinusoid that we sample at discrete time points n has the following form and identity:

$$\begin{aligned} s_n &= A \sin(\omega n + \phi) \\ s_n &= \frac{1}{\cos \omega} \left[\frac{s_{n-1} + s_{n+1}}{2} \right] \end{aligned}$$

This allows, as in [1], us to define an error function

$$\mathcal{E}(\alpha) = \sum_n \left[x_n - \alpha \left(\frac{x_{n-1} + x_{n+1}}{2} \right) \right]^2 .$$

If our signal is well described by a sinusoid, then when $\alpha = (\cos \omega)^{-1}$, the error should be small. The solution to our least squares is given by

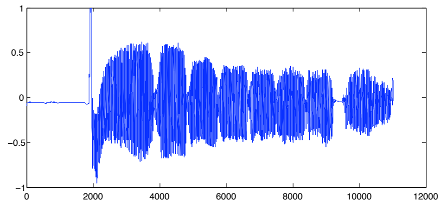
$$\alpha^* = \frac{2 \sum_n x_n (x_{n-1} + x_{n+1})}{\sum_n (x_{n-1} + x_{n+1})^2}.$$

Thus, we minimize our signal's error function and then check that our signal is sinusoidal rather than exponential and not zero. Then our estimated frequency is

$$\omega^* = \cos^{-1}(1/\alpha^*)$$

3.2 Detecting Pitch in Speech

In our implementation, we followed Saul et al.'s approach of running our sung query signals through a low pass filter to remove high frequency noise, using half-wave rectification to remove negative energy and concentrate it at the fundamental, then separating our signals into a series of eight bands using bandpass 8th order Chebyshev filters. We can then use Prony's method for our sinusoid detection, which has proven accurate in previous tests [1]. Saul et al. also define cost functions that allow us to determine whether sounds are voiced or not and whether the least squares method has provided an accurate enough fit to a sinusoid (see [1] for more details).


(a) Waveform of Scale

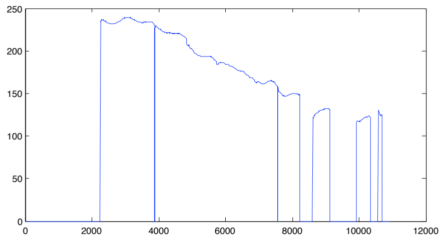

(b) Frequencies of Sung Scale

Figure 1: Raw Data and Frequencies

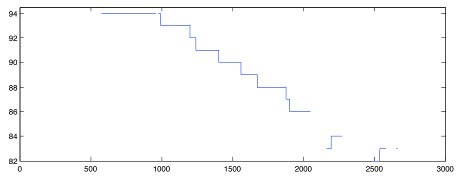


Figure 2: Pitches of the Sung Scale

3.3 Transforming to Melody

Using the above method, we retrieve a frequency at every .04 second interval, which we downsample from 22050 Hz to 918 Hz, which allows for quicker computations. Given our set of frequencies $\{f_0, \ldots, f_n\}$ over our n samples, we assign each f_i to its corresponding MIDI pitch $p_i \in [0, 127]$, then use mean smoothing to achieve better pitch estimates for every p_i. We group consecutive identical pitches from the samples to give us our melody $\langle \mathbf{p}, \mathbf{d} \rangle = \langle (p_1, d_1), \dots, (p_k, d_k) \rangle$. Lastly, we compress this melody to be in a 12 note (one octave) range, because it helps our computational complexity in the alignment part of our algorithm to have fewer possible pitches, and spectra alignments are not overly sensitive to octave-off errors. To see examples of frequencies and pitches extracted from singing, see figures 1 and 2.

4 A Generative Model from Melodies to Signals

As mentioned in section 2, we have a generative model that we are trying to maximize: $P(\mathbf{o}|\mathbf{p}, \mathbf{d})$. More concretely, given a melody query $\langle \mathbf{p}, \mathbf{d} \rangle$, we would like to find the acoustic performance $\mathbf{o}$ that $\langle \mathbf{p}, \mathbf{d} \rangle$ is most likely to have generated.

4.1 Probabilistic Time Scaling

As in [2], we treat the tempo sequence as independent of the melody (which ought to hold for short pieces), to give us problem of finding the $\mathbf{o}$ in our database that maximizes the following:

$$P(\mathbf{o}|\mathbf{p}, \mathbf{d}) = \sum_m P(\mathbf{m}) P(\mathbf{o}|\mathbf{p}, \mathbf{d}, \mathbf{m}).$$

In this, having the $\mathbf{m}$ parameter in the conditional simply means that we are scaling the sequence of durations $\mathbf{d}$ by $\mathbf{m}$. $\mathbf{m}$ is modeled as a first order Markov process, so

$$P(\mathbf{m}) = P(m_1) \prod_{i=2}^{k} P(m_i | m_{i-1}).$$

Because the log-normal distribution has the nice trait that it somewhat accurately reflects a person's tendency to speed up (rather than slow down) when doing a musical query or performance, we say that

$$P(m_i | m_{i-1}) = \frac{1}{\sqrt{2\pi}\rho} e^{-\frac{1}{2\rho^2}(\log \frac{m_i}{m_{i-1}})^2}.$$

We also assume a log-normal distribution of $P(m_1)$, so $\log_2(m_1) \sim \mathcal{N}(0, \rho)$. In these equations, the ρ parameter describes how sensitive our model is to local tempo changes–high ρ ($\rho > 10$) means that our model is not very sensitive to tempo changes, low ρ ($\rho < 1$) give us a model very sensitive to tempo changes.

4.2 Modeling Spectral Distribution

We let $\bar{o}_i$ represent a sequence of samples (that we suppose is generated by note and duration (p_i, d_i) in our query) from a piece in our database. That is, $\bar{o}_i = o_{t'+1}, \dots o_t$, where p_i ends at time sample t and $t' = t - d_i$. We use a harmonic model of $P(\bar{o}_i)$ almost identical to that in [2].

$F(\bar{o}_i)$ is the observed energy of some block of samples $\bar{o}_i$ over the entire spectra (we get this from the Fourier transform). Also, we assume that we have a soloist in all of our recordings, and that $S(\omega, \bar{o}_i)$ is the energy of the soloist at frequency ω for our samples, and our model assumes that S is simply bursts of energy centered at the harmonics of some pitch p_i. This is a reasonable assumption for our soloists energy, because often the harmonics of the accompaniment will roughly follow the soloist. That is, we have a burst at $p_i h$ for $h \in \{1, 2, \dots, H\}$, and we set H to be 20 to keep the number of harmonics reasonable. We can define the noise of a signal at some frequency ω to be the energy that is not in the soloist or any of his or her harmonics (frequencies that are multiples of ω), or $N(\omega, \bar{o}_i) = F(\bar{o}_i) - S(\omega, \bar{o}_i)$. This gives us that

$$\log P(\bar{o}_i | p_i, d_i) \propto \log \frac{||S(\omega, \bar{o}_i)||^2}{||N(\omega, \bar{o}_i)||^2}$$

where $|| \cdot ||$ is the l_2-norm (see [2] for this derivation). We assume that d_i is implicit in conditional probabilities when given p_i from here on, because they pitches and durations in our queries come in pairs.

To actually get the energy of the soloist and the noise, assuming the soloist is performing at a frequency ω, we use a method called subharmonic summation proposed by Hermes [7]. This method allows us to determine if a pitch is predominant in a spectrum by adding all the amplitudes of its harmonics to the fundamental frequency. The formula we apply is as follows:

$$S(\omega) = \sum_{h=1}^{H} d^h F(h\omega)$$

where d is a contraction rate that usually is set to make it so that lower frequencies are more important (we set $d = 1$ so we can simply remove all energy at the frequencies we assume are the soloist's). Thus, when we are performing a query of a piece and we would like find the probability of a block of signals $\bar{o}_i$ given the current pitch in our query, we simply remove all the peak frequencies at multiples of our query's pitch's frequency, then find the remaining signals and treat them as noise (see figure 3). This gives us $P(\bar{o}_i | p_i)$.

4.3 Matching Algorithm

With the background we have now put in place, we see we can develop a dynamic programming algorithm as in

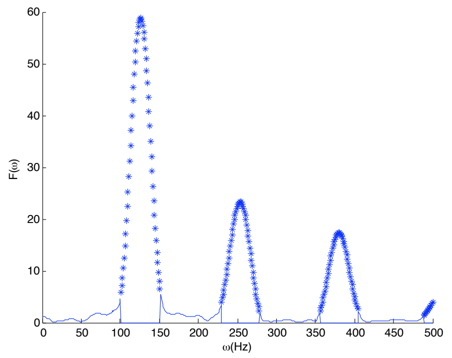


Figure 3: Solo vs. Noise. Stars are solo frequency, the rest is noise

[2] to retrieve our polyphonic piece given some k-length query of pitch-duration pairs $\langle (p_1, d_1), \ldots, (p_k, d_k) \rangle$.

More specifically, in our implementation, for a given polyphonic piece, we have a spectrum sample every .04 seconds, and there are T samples for the entire length of the piece. Recall that $P(\bar{o}_i|p_i) = P(o_{t'+1}, \ldots, o_t|p_i)$ for appropriate t, t'. We also have the tempo scaling factors to account for, that is, $P(m_i|m_{i-1})$ from above. In the algorithm, we call these tempo scaling factors ξ (to vary tempo for our algorithm, each scaling factor is simply a different small multiple of .04 that is added or subtracted from d_i to give us a different duration). There is also a chance that there are rests in the pieces we consider, and we must take into account rests in our queries. As such, if we have that $p_i = 0$, we replace $P(\bar{o}_i|p_i)$ with the spectrum probability of a rest

$$\begin{aligned} P(\text{Rest}|p_i = 0) = \\ \frac{1}{2}(2 - P(o_{t'+1}, \ldots, o_t|p_{i-1}) \\ -P(o_{t'+1}, \ldots, o_t|p_{i+1})) \end{aligned}$$

In our model, this says that if we have a rest in our query, then the pitches before and after pitch p_i in our query ought not be very present in the spectrum.

Putting all of this together, we define $\gamma(i, t, \xi)$ to be the joint likelihood of $\mathbf{o}^t$ and $\mathbf{m}^i$, or as the maximum (over the set M of all the scaling factors) probability that the i^{th} note of our query ends at sample index t, and its duration is scaled by ξ.

$$\gamma(i, t, \xi) = \max_{\mathbf{m^i} \in M^i} P(\mathbf{o}^t, \mathbf{m}^i | \mathbf{p}, \mathbf{d})$$

While this is the joint likelihood of our polyphonic piece's first t samples and its first i scaling factors given $\mathbf{p}$ and $\mathbf{d}$, and ideally we would have just the likelihood of the samples $\mathbf{o} = o_1, \ldots, o_T$, we still use γ as the retrieval score given our query. All this gives us the alignment algorithm (reminiscent of most probable path algorithms for HMMs) that we see in figure 4, due to [2] with some modifications.

1. **Initialization**

$$\forall t, 1 \leq t \leq T, \gamma(0, t, 1) = 1$$

2. **Inductive Building of γ**
   for $i := 1$ to k, $t := 0$ to T, $\xi := \min \xi$ to $\max \xi$

$$\begin{aligned} \gamma(i, t, \xi) = \\ \max_{\xi' \in M} \gamma(i-1, t', \xi') P(\xi|\xi') \cdot P(o_{t'+1}, \ldots, o_t|p_i) \end{aligned}$$

   where $t' = t - (d_i + \xi)$.

3. **Termination**

$$P^* = \max_{1 \leq t \leq T, \xi \in M} \gamma(k, t, \xi)$$

Figure 4: The alignment algorithm we use

4.4 Complexity of Matching a Query

The complexity of this algorithm, which is relatively easy to see from the for loop nesting, is $O(kT|M|^2)$, where k is the number of notes in our query, T is the number of time samples in the polyphonic piece we query, and M is the set of possible tempo scaling values. This holds as long as $P(o_{t'+1}, \ldots, o_t|p_i)$ can be computed in constant time, which we guarantee in our implementation.

To achieve constant time probability lookups, we pre-compute all the probability values of sample blocks $\bar{o}_i$ using fast Fourier transforms with 2^{15} points for good resolution. We compute the probability $P(\bar{o}|p)$ for each pitch p that we can see in our queries for all the possible lengths of samples in our audio signal $\mathbf{o}$. We compute probability for every block of samples $o_t \ldots, o_{t'}$ of length .04 to 2.5 seconds, because singers cannot change pitch in under .04s, and in most music, especially the music we use, pitches are rarely held for longer than two seconds. Effectively, this gives us $O(T \cdot 62)$ probabilities for each pitch, of which we have 12. This pre-computation, while expensive, significantly helps running times, because we do not have to do spectral analyses every time we wish to calculate $P(\bar{o}|p)$ in our algorithm.

5 Experimental Results

We ran tests on five different Beatles songs–*Hey Jude*, *Let It Be*, *Yesterday*, *It's Only Love*, and *Ticket to Ride*. The system, given a melody represented by pitch-duration pairs, retrieved the song whose alignment score was the highest. As a first test, the system was given correct symbolic representations of parts of all five songs, copied directly from scores. In this, the retrieval was perfect, as expected for our small database.

5.1 Sung Queries in Key

Our system's retrieval rates on the five songs, given queries that were sung in the song's keys, were again perfect. The average ratio

$$\frac{\text{Highest retrieval score for query } \langle \mathbf{p}, \mathbf{d} \rangle}{\text{Second highest retrieval score for query } \langle \mathbf{p}, \mathbf{d} \rangle}$$

was .23 for queries sung in the correct key. This accuracy is fairly good, though it is orders of magnitude worse than the accuracy achieved with correct symbolic queries. Thus, as long as the system did not have to do any transposition, the querying worked for our small database.

5.2 Transposition

To test our system's resilience to transposition (shifting an entire melody but keeping its relative pitches constant), we had the alignment procedure attempt to align the melody we gave it, as well as the 11 other melodies that were transpositions (the i^{th} transposition simply shifts all the pitches of $\mathbf{p}$ up i) of the original melody. The piece retrieved was the one which had the maximum alignment score on any one transposition of the melody.

As before, when given correct symbolic representations (transposed scores), the retrieval procedure was flawless, always returning correct results.

When the queries, however, were sung but not in the key in which the Beatles sang (for example, *Yesterday* sung in E instead of F, a half step down), results were not as optimal. *Hey Jude* and *It's Only Love* the system still identified correctly, but the other three songs had significantly worse results, sometimes being given lower alignment scores on a melody than as many as three other songs. The reasons for this are not totally clear, but we speculate that transpositions of a query may put it into the key of a different song from our database, which would make it easier to for a query to match the spectra of an incorrect song.

6 Conclusions and Future

We have taken a step toward building a polyphonic music database that can be queried by singing. While we met with success as long as queries were in the correct key, the system's inability to handle transposition is bothersome and will be a subject of future work. We also would like to expand the system to handle incorrect accidental modulations in singing, to give it a distribution over incorrect pitches. In the inductive part of the algorithm, instead of taking $P(\bar{o}|p)$, we could define a distribution over the probability that the user meant to sing note p in his query. For example, we might look at $p-1, p, p+1$ and take the maximum of $\{\frac{1}{2}P(\bar{o}|p-1), P(\bar{o}|p), \frac{1}{2}P(\bar{o}|p+1)\}$, which would allow the singer to miss some pitches by a semitone but would increase the time complexity of our algorithm by a factor of the number of pitches over which we take a distribution to allow incorrect singing.

The system's speed is also relatively low; to build a large database the alignment procedure would need a significant speedup. It may be useful to look into learning to automatically extract themes from polyphonic music, then performing queries over those themes. In spite of the difficulties inherent in this problem, we have demonstrated that a query by humming system searching polyphonic audio tracks is feasible.

References

[1] Saul, L., Lee, D., Isbell, C., and LeCun, E. "Real Time Voice Processing with Audiovisual Feedback: Toward Autonomous Agents with Perfect Pitch," *Advances in Neural Information Processing Systems 15*, pp. 1205-1212, MIT Press: Cambridge, MA, 2003.

[2] Shalev-Shwartz, S., Dubnov, S., Friedman, N., and Singer, Y. "Robust Temporal and Spectral Modeling for Query by Melody," *SIGIR02*, pp. 331-338, ACM Press: New York, NY, 2002.

[3] Proakis, J., Rader, C., Ling, F., Nikias, C., Moonen, M., and Proudler, I. *Algorithms for Statistical Signal Processing*, Prentice Hall, 2002.

[4] Rabiner, L. "On the Use of Autocorrelation Analysis for Pitch Determination," *IEEE Transactions on Acoustics, Speech, and Signal Processing, 25*, pp. 22-33, 1977.

[5] Meek, C. and Birmingham, W. "Johnny Can't Sing: A Comprehensive Error Model for Sung Music Queries," in *Proc. ISMIR*, 2002.

[6] Durey, A. and Clements, M. "Melody Spotting Using Hidden Markov Models," in *Proc. ISMIR*, 2001.

[7] Hermes, D. "Measurement of Pitch by Subharmonic Summation," *Journal of Acoustical Society of America, 83(1)*, pp. 257-263, 1988.