# Project 5: MySong

*Due: 12:00 PM (Noon!), May 11, 2009*

## Contents

**1 Introduction** **1**

**2 Music 101** **2**

**3 The HMM** **2**

**4 What to Do** **3**

**5 Notes** **4**

**6 Handing In** **5**

**7 Support Code** **5**

**8 Resources** **6**

## 1 Introduction

Microsoft SongSmith (formerly MySong) is a research project led in part by Dan Morris ('00). Here is an overview of MySong, excerpted the MySong website:

> MySong, introduced in our CHI 2008 paper, automatically chooses chords to accompany a vocal melody, allowing a user with no musical training to rapidly create accompanied music. MySong is a creative tool for folks who like to sing but would never get a chance to experiment with creating real original music. Come on, you know who you are. . . you sing in the car, or in the shower, or you go to karaoke clubs, or you just once in a while

find yourself singing along with catchy commercial jingles. MySong is also a great tool for songwriters who want to quickly experiment with melodies and accompaniments.

It turns out that the technology underlying MySong are a couple of relatively straightforward HMMs, in which chords are states and notes are observations. Using this model, your goal in this project is to implement the Viterbi algorithm to decode a melody (i.e., a sequence of notes) into a chord progression. That is, you will build your own MySong (*a.k.a*, YourSong).

## 2 Music 101

In order to understand this write-up, you'll need to know some basic music terminology and concepts.

- In western music, the scale consists of 12 musical notes: {`C, C♯, D, D♯, E, F, F♯, G, G♯, A, A♯, B`}. A *melody* is a sequence of notes. For example, the melody *Twinkle Twinkle Little Star* can be written as the sequence of notes `C C G G A A G F F E E D D C`.
- A *measure* is a natural division in a song, determined by the tempo. For example, most pop songs are divided into measures of four beats each. Dividing *Twinkle Twinkle Little Star* into measures of four beats each yields {`C C G G | A A G | F F E E | D D C`}, with `|` denoting a division between measures. The `G` in the second measure and `C` in the fourth measure are held for two beats; every other note is held for one beat.
- A *chord* is a harmonic structure that accompanies a melody. In the MySong universe, chords come in five different flavors: {`Major, Minor, Diminished, Augmented, Suspended`}. Each chord consists of several notes, built off one `Root`. For example, a `C Major` chord is composed of the notes `C, E, G`, while a `D Major` chord is composed of the notes `D, F♯, A`. As there are 12 notes and 5 types of chords, there are a total of 60 chords in the MySong universe.
- Lastly, a song's *key* is its melodic and harmonic "center." For example, a song in the key of `C Major` is more likely to start and end with a `C Major` chord. In the MySong universe, there are 24 keys: {`C, C♯, ..., B`} `Major` and {`C, C♯, ..., B`} `Minor`.

*You do not need to know any music theory beyond the basic definitions given above to do this project.*

## 3 The HMM

Recall that a hidden Markov model (HMM) is a tuple $H = \langle X, Y, \Pi, A, B \rangle$, where $X$ is a finite set of states, $Y$ is a finite alphabet of observations, $\Pi$ is an initial probability distribution over states, $A$ is a transition probability matrix, and $B$ is an emission probability matrix. For this project, we extend the definition of an HMM to also include $\Psi$, a final probability distribution over states.

As we noted above, in the MySong HMMs, chords are states and notes are observations. Your goal in this project is to write a program that will take as input a melody, that is, a sequence of notes, and generate a corresponding sequence of backup chords. More specifically, *your program should use the Viterbi algorithm to compute the most likely chord sequence, given a melody.*

Here are some more details about the MySong HMMs.

- $X$ is the set of (60) possible chords. For convenience, each chord is represented by a single integer, 0 through 59. The mapping from integers to chords follows the usual enumeration order: i.e. `0 = C Major, 1 = C Minor, 2 = C Diminished, 3 = C Augmented, 4 = C Suspended, 5 = C♯ Major, ..., 59 = B Suspended`.
- $Y$ is the set of (12) possible notes. Each note is represented by its pitch, or frequency. For example, on a piano, the `A` above middle `C` is usually tuned to 440 Hertz.
- The parameters of the MySong HMMs (i.e., $\Pi$, $\Psi$, $A$, and $B$) were provided by Dan Morris ('00). He and his team at Microsoft built these matrices by analyzing approximately 300 lead sheets, which describe melodies and the chord changes that accompany them. They divided these lead sheets into two sets—songs in major keys and songs in minor keys—and they built two corresponding HMMs. The result is two initial probability distributions (one major and one minor); two final probability distributions (one major and one minor); and two transition probability matrices (one major and one minor). Note that there is only one emission probability matrix; this matrix models the probability of a note given a chord, a property which is independent of key.
  - Each transition probability matrix $A$ has dimensions $60 \times 60$, with each entry reporting the probability of transitioning from one chord to another. These values are simply the number of times that each transition was observed in the corresponding set of lead sheets, normalized to convert from counts to probabilities.
  - Each initial probability distribution $\Pi$ is a 60 row by 1 column matrix, where each entry indicates the probability that a song initiates in that state. Similarly, the final probability distribution $\Psi$ is a 60 row by 1 column matrix, where each entry indicates the probability that a song terminates in that state.
  - Finally, the emission probability matrix $B$ is a 60 row by 12 column matrix, where each row is a probability distribution over notes. This distribution represents the notes that tend to be played over the chord represented by the row (irrespective of key).

Here are some important details about the way in which we represent a melody, the input to your program:

- By definition, a melody is a sequence of notes, and a note is a pitch. But rather than model a melody as a sequence of pitches, we model a melody as a sequence of *histograms* of pitches, where each histogram corresponds to one measure. More specifically, each histogram recounts the number of centiseconds during which each pitch was observed during each measure (see Figure 1). Correspondingly, you should associate only one chord with each histogram. That is, your Viterbi implementation should find the single best chord to back up each measure.

## 4 What to Do

1. Implement Viterbi to decode a melody in which the observation corresponding to each measure is reduced from a histogram of pitches to a single representative pitch. How you determine this note/pitch is up to you.

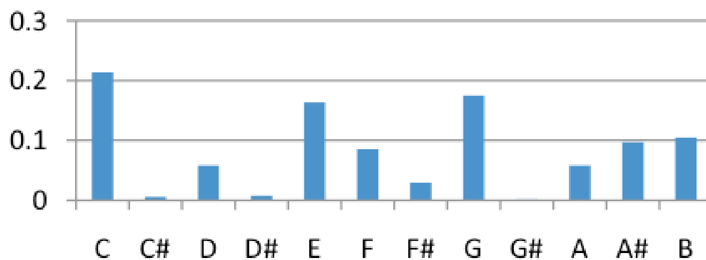


Figure 1: A graphical illustration of a single observed measure (normalized). The notes `C`, `E`, and `G` occurred most frequently during this measure.

2. The usual implementation of Viterbi applies to a sequence of observations $y^{1:T}$ in which each $y^t \in Y$. The goal of this problem is to extend Viterbi to apply to a sequence of observations $y^{1:T}$ in which each $y^t \in \Delta(Y)$, the set of probability distributions over the set $Y$. More specifically, you should extend your implementation of Viterbi to decode a melody in which the observations corresponding to each measure are full histograms. To do so, instead of making use of only $b_k$, the $k$th column in the emission probability matrix, as the usual Viterbi algorithm does whenever $y^t = k \in Y$, you should find another way to reduce the emission probability matrix to a single column—a way that incorporates all the available information about the observation $y^t \in \Delta(Y)$ (not just a single statistic such as the mode).

3. For extra credit, you can play around with the various probability matrices and/or with the Viterbi algorithm itself to see how small changes impact the chord progressions output by your program. If your chord progressions become more interesting after tweaking the probabilities and/or the algorithm, document your method. We plan to relay successful methods to Dan. Your method could be incorporated into the next release of SongSmith!

# 5 Notes

The transition, start, and end probability matrices are specific to the key `C Major` for the major model and `A Minor` for the minor model. This is why $\Pi$ is peaked at chord 0 (`C Major`). A song in `C Major` is more likely to start on a `C Major` chord. However, not all of the melodies given are in `C Major`; in fact, most are not. Hence, for each model (major or minor), you should run the Viterbi algorithm 12 times, once per key, and then return the path with the highest probability.

In order to run Viterbi in each major key, you will need to change how you index the $\Pi$, $\Psi$, and $A$ matrices, as they were all built for `C Major`. For instance, in order to treat the $\Pi$ matrix as if it is in the key of `C♯ Major`, you will need to offset the index of each chord by $-5$. That way, the start probability of the fifth chord, `C♯ Major`, will be set to the value $\pi_0$. The same type of shift in indexing is required for the minor model, where all probability matrices are specific to `A Minor`.

**Log Probabilities** Underflow is common when multiplying small probabilities, so you may find it preferable to add the logarithms of probabilities, rather than multiply the probabilities themselves. This trick is simple to apply to Viterbi, where the two only operations are multiplication and

maximization. It is more difficult to apply to the the forward and backward algorithms (to compute $\alpha$ and $\beta$), because there the two operations are multiplication and addition.

## 6 Handing In

Please hand in the following:

1. All the code necessary to solve the assignment (including our support code).
2. A README file that includes the following:
   (a) Standard bug reports.
   (b) Anything that might help us read your code: design quirks, conventions, etc.
   (c) Comments that might help us improve and/or extend this project in the future.
3. A short report of how your second implementation worked, and how it differed from the first.

Hand in by typing `cs141_handin mysong` in the shell from the directory containing your work.

## 7 Support Code

As usual, install the support code by running the install script:

```
/course/cs141/bin/cs141install mysong
```

The support code will be installed into your `/course/cs141/projects/` directory.

The support code is split into three packages: `input, mysong` and `player`.

- The `input` package : The `PitchTracker` takes in a `.wav` file, and returns a vector of `Measure` objects. The `ModelParser` has two static methods, `transitionModel()` and `observationModel()`. The former returns a `TransitionModel` object (that contains the transition probability matrix $A$, as well as the start, $\Pi$, and end $\Psi$ probability matrices), while the latter returns the emission probability matrix, $B$. The `transitionModel` method takes as input a boolean: `true` for the major model, and `false` for the minor model.
- The `MySong` package : This package includes `MySongMain`, the only class you will need to add code to. In this class, you will invoke the input class to get the matrices and the melody information, and you will implement Viterbi. This package also includes the `Chord` and `Measure` objects. The `Measure` object maintains the normalized histogram of pitches, in an array of 12 doubles called `normHist`.
- The `Player` package : The `MidiWavMixer` takes in a `.wav` file path, a vector of `Chords`, the number of beats per minute and the number of beats per measure. It plays the `.wav` file and the chord progression simultaneously. If they appear to be off, there are two parameters you can play around with to try to sync them up. You can see where they are in the comments of the `MidiPlayer` class. If the midi piano is too loud or soft, you can change the `VELOCITY` parameter in the `MidiPlayer` class (this is basically a value for the volume of the `Player`).

The file `pitchtrack.praat` will be installed with the stencil code. This must be in your outer most `mysong` directory, as it is the script that is called by the `PitchTracker`.

We recommend working with eclipse for this project. Be sure to make the `mysong` folder the project folder, so Java knows where to find the `pitchtrack.praat` file.

A `models` folder will also be installed with your code. This includes the major transition model, minor transition model, and the observation model (i.e., the emission probability matrix). *You need not touch these.* But if you accidentally delete one or more of them, a copy of the folder lives in `/course/cs141/lib/mysonglib/models`.

The `.wav` files available as inputs to your program are located in `/course/cs141/lib/mysong/wavfiles`. The filenames are formatted like this: "filename.120.wav", where the number is the tempo (in beats per minute) of the melody. All provided files have four beats per measure.

**Note:** To hear the the input to and the output of your program, you will need to plug headphones into the back of your computer. In order to change the volume on the department machines, type `gnome-volume-control` into a shell. This will give you a volume control interface for the sound card. You can test your sound by typing `play <wavfile>` into a shell. If your sound is not working, a consultant can fix it for you. This is simply an ownership issue (if someone logged in on your machine before you, he or she owns the sound card, and you cannot use it).

## 8 Resources

1. First MySong Paper:
   `http://research.microsoft.com/en-us/um/people/dan/mysong/MySongCHI2008.pdf`
2. Second MySong Paper:
   `http://research.microsoft.com/en-us/um/people/dan/mysong/MySongAAAI2008.pdf`
3. Mysong Homepage:
   `http://research.microsoft.com/en-us/um/people/dan/mysong/`
4. Viterbi Algorithm on Wikipedia:
   `http://en.wikipedia.org/wiki/Viterbi_algorithm`