CS 221 Final Project Proposal: The Canon Solver

Bryan McCann (bmccann) & Zachary Yellin-Flaherty (zachyf)

Project Description

The ultimate goal of our project is to solve composer puzzles in which a composer offers a single melody, and the solution is a multi-voice composition involving that melody and translations of the melody as other parts. Options for translation typically include staggering the melody (time-shifting), transposing it, or reversing it.

Problem Statement -- Canons as Search Problems

Input:

Melodies are represented as a comma-separated string of (start-time, pitch, duration, key signature, time signature) tuples. These strings are translated into vectors with one index per lowest note denomination. Entries of the vector correspond to the number representation (MIDI number) of the note being played on that beat.

Output:

1. One of the most harmonious (least costly) states as measured by the cost (dissonance) of overlaying the state on the original melody.
2. One of the states that makes the next layer the most costly (for competitive environments).
3. A round with as many parts as possible without losing harmony.

Successor States:

Possible neighboring states result from time-shifting, transposing, reversing, or some combination of those actions on the original melody. Time-shifting corresponds to shifting the vector and wrapping the excess around to the front of the vector. Transposing is adding a constant factor to each note to create harmony with the original melody (For instance, adding a second voice up a 3rd or 6th is an often-employed harmonization technique. Special care will be taken to make sure we select the correct major or minor 3rd or 6th depending on the key signature). Reversal simply reverses the order of the notes in the original melody.

Costs:

Costs are defined between the original vector melody and neighboring state vectors. The cost of a neighboring state is the sum of the costs calculated for each interval in the vector where perfect intervals are the least costly (most harmonious), followed by consonant intervals (major/minor 3rd/6th), and finally dissonant intervals (2nd, 7th, diminished, augmented).

Baseline Testing

In our baseline testing we framed the problem as a search problem and used uniform cost search to construct a simple two-part round. We represented the song “Row, Row, Row Your Boat”, and then asked our algorithm to define the best second voice based on the costs of consonance and dissonance. The result, which you can hear in the attached mp4, is quite harmonic. While it didn’t assign the highest score to the second voice that we sang in elementary school, the result it chose was the most harmonic in terms of the attributes we gave it.

We simplified our problem statement and model for baseline testing by using a smaller set of actions (only time-shifting) and using a simplified state (pitch, duration). Later iterations will include an abstraction of this system: representation of chords in general rather than single notes, adding more voices by making our state a matrix rather than a vector, and allowing for transposition and reversal through row/column manipulation on the matrix, and more nuanced costs:

- Representing triads and chords will expand the size of our search space, but chords are finite and can be easily represented by a series of distances between its constituent voices.

- Adding more than one voice will allow us to reach our second and third desired outputs.

- Adding transposition and reversal will also increase the possible number of states, but these actions will be necessary to tackle the more complex puzzles created by Bach and other classical composers.

- Timing (in our simple example, we allowed parts to enter mid-bar or mid-beat with no cost), and the order of chords represented by ‘bigram’ features will make our algorithm more effective. For example, with multiple voices, a subdominant chord will most likely be followed by a dominant chord, and we can penalize retrogression, and similarly, even in two part harmonies we can assign lower cost to tension followed by resolution and common cadences than non-standard progressions. Lastly, we might apply penalties for choosing notes that are simply far away from each other on a keyboard, representing a penalties as drawn from a Gaussian distribution.

Notably, the simple song we chose will have more options for harmonious voices than more melodically complex pieces. Since all notes are diationic to the major key (in our case C major), many of the choices scored in a similar range. With more complex melodies, including Bach's puzzles that we will tackle later, unintended melodies will be more blatantly incorrect, and they will yield fewer acceptable outputs.

Possible Extension -- Stylistic Learning

For our simple task, we arbitrarily assigned perfect intervals a cost of 0, other consonant harmonic intervals a cost of 0.2, and dissonant intervals a cost of 0.4. With training data, we can experiment with different learning algorithms to tune these parameters.

For the next step, we would use the UCI Machine Learning Bach Chorales dataset, which we cleaned but did not use. We would tune our parameters to solve the musical puzzles in the style of Bach.