Eli Stine & Nick Towbin-Jones, AI Final Project Final Report, 5/6/13

Trained/"Trane'd" Music Improvisation Generator (TMIG)

Introduction:

This project aimed to generate improvisations (specifically jazz solos over chords) in the style of a performer, given transcriptions and accompanying chords of real performances as training data. The program we created consists of two main parts: building the models that represent the training data and generating "improvised" MIDI (Musical Instrument Digital Interface, a protocol for representing musical notes in computers) output given a formatted list of chords.

This project differs itself from comparable programs in that it uses transcriptions of real performers to generate its "improvisations" (rather than using properties of chords, for example (see GenJam in appendix)), tightly associates training data with chords, and offers a good deal of flexibility (for example, models can be extended with as many different songs as a user would like, and output can be on any (satisfiable) chord list).

We found that while test participants were not always 100% sure of the "fakeness" of our generated "improvisations", they were more likely to accurately classify a longer example than a shorter one, revealing that our system can sound genuine on a small scale, but long-term phrasing and melodic contour needs work.

Problem Definition and Algorithm:

Task Definition

Input: This program takes in monophonic MIDI file(s) of an improvisation based on a set of chords and formatted list(s) of chords that correspond to the MIDI file.

Output: Given another formatted list of chords (either the same or different than the input(s)) new monophonic MIDI file(s) are generated which use the data collected/learned in the input step to "improvise" over the chords.

This simple 2-step process offers more flexibility than it seems: as mentioned earlier, multiple songs may be used to train the algorithm, and chords from any one of them can be "improvised" on. Possible uses include aiding a jazz performer who is interested in seeing how a set of musical ideas can be spun out over time or giving a composer different, randomly generated examples of how a set of musical materials associated with chords could be manipulated in a piece they are writing.

Algorithm Definition

1. Command line arguments for inputmidifile, inputchordlist, and order are used to build Models

    1a. The notes in the MIDI file are parsed out as a list of Note objects

    1b. The chords in the chord list are parsed out as a list of Chord objs.

1c. Each pair of chords is made into a ChordPair object which is associated with the notes that occur in the first chord of the chord pair

1d. Each tuple of notes (length = order) in the ChordPair is associated with the note that follows it

1e. Each following note is associated with how many times it occurs as a note following the tuple in all of the input

2. The user is then prompted for any other MIDI file + chord list pairs they want to enter, which extend the Models created in step 1

3. The user is then prompted for a formatted output chord list, which is used to generate the "improvisation":

3a. The chords in the chord list are parsed out as a list of Chord objs.

3b. The model is first checked with each chord to make sure that each chord can be satisfied by the model. An error will be returned to the user if any one of the chord pairs in the input cannot be satisfied.

3c. For each 2 chords we create a ChordPair, and poll our Models map for the value of this ChordPair

3d. From the Markov Model returned we get a weighted random tuple of previous notes (higher probability for tuples that have more following notes), which we save in our output

3e. With the previous notes from 3d, we get a weighted random following note

3f. We add the following note to our output (and reduce the empty space in the current chord by its length), and adjust previous notes to include this note (shifting one other note out)

3g. While we still have space in the chord, we continue adding notes this way. Once we have run out of space, we get a new ChordPair, until there are no more chords. We associate the last chord with the first chord (many jazz songs are cyclical)

4. The "improvised" output is parsed to a MIDI file and written to disk

System Design

We wrote our program in Java, primarily because it had easy-to-use MIDI capabilities.

We began by concisely representing Notes, Chords, and Chord Pairs. Notes contain their pitch (an integer, as in MIDI) and length (quarter note = 1). Chords contain their pitch (abstracted pitch class, with 0 representing "C"), type (minor, major, or dominant), and length. ChordPairs contain an int that represents the relationship between the current and next chord in the chord pair, along with the type of the current and next chord (note that there is no "length" parameter).

The primary reason for defining the Notes, Chords, and ChordPairs in this way was to make the relationship between notes and chords as generic as possible. For example, as ChordPairs are defined not by the pitches of the current and next chord but by their relationship/distance, a ChordPair for C -> F (relationship: +5) can be applied to the same chord relationship in a different key (for example, C# -> F# (1 -> 6), but still +5 relationship). Along the same lines, all Notes are pitch-shifted to be relative to the key of the chord they are in. This means that when a specific ChordPair is asked for in the output generating phase (D -> G (2 -> 7), for example) the relationship between the chords can be used to get a generic ChordPair from our constructed model (relationship: +5). The notes associated with this ChordPair are then pitch shifted to "work" inside of the key of the specific ChordPair (2 -> 7 in our case, so we pitch shift the Notes by +2).

Next we constructed parsers and a MIDI writer. The parsers take our MIDI input and formatted chord lists and parse them into lists of Notes and lists of Chords, respectively. Several other pieces of information must be parsed from the MIDI file, including resolution (so we know what time value, in ticks, the length of a quarter note is in the file) and BPM (beats per minute, the tempo of the file, which we use when outputting). The MIDI writer takes a list of Notes and a tempo and writes a MIDI file to disk.

We then began the main functionality of the program, which uses our data types, parsers, and writers. The TMIG class functions as the interface between the user and all of the functionality of our program, taking command line arguments for input MIDI file, input chord list, and Markov model order. The TMIG class contains 2 crucial methods: buildModels and generateOutput. These methods handle the higher level looping and method-calling required for model building and "improvisation" generation.

The real workhorse of our system is 3 layers of maps:

1. "Models", which contains a hashmap from ChordPairs -> MarkovModels

2. "MarkovModel", which contains a hashmap from tuples of Notes -> Markovs

3. "Markov", which contains a hashmap from Notes -> How often they occur as notes following the associated tuple

Markov and MarkovModel contain generateRandom() methods, which return notes and tuples weighted by how frequently they occur following a given tuple or how many notes follow them, respectively. Models functions as an extra layer in order to provide the correct MarkovModel for a given ChordPair, and contains high level parsing and generating functions.

Experimental Evaluation

Methodology

We decided to test our program by having people listen to examples of the output of our system interspersed with real live performances (both with the same fidelity and accompaniment) and then rate how confident they are that they are computer generated or a genuine performance, along with some contextual questions.

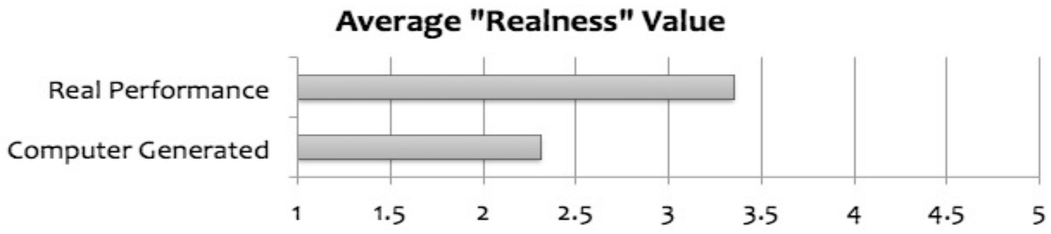
Specifically our test asked the following:

* 1. How familiar are you with Coltrane's "Giant Steps" and "Lazy Bird"?
    * 1 = "never heard of", 5 = "familiar with recordings"
* 2. Musical experience
    * 1 = "no musical experience", 5 = "experienced musician"
* 3. Rate confidence that each of the 8 audio examples are:
    * 1 = "computer generated", 5 = "real live performance"

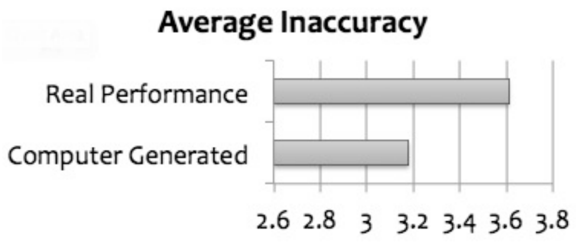
The tests were conducted with headphones in a space with no extraneous musical sound. Examples were separated by 2 seconds of silence, and participants were asked to NOT re-listen to an example, but were allowed to pause in between them.

Results

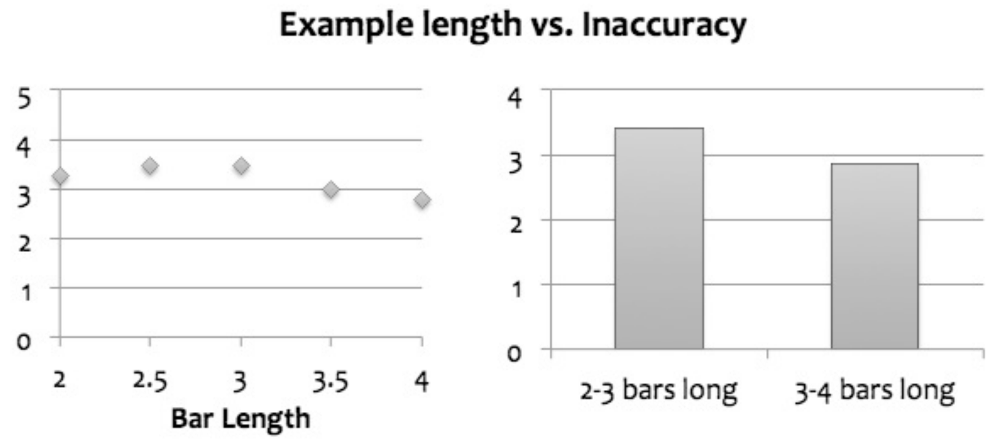
We ended up testing 9 people, which amounts to 72 examples listened to in 2 tests, "A" and "B" versions (each of which contained 4 computer generated and 4 real performance examples). We threw out the first datum of each test, as this "situates" the listener, giving them a reference to work from. Of the 16 examples listened to in the tests, 9 were "short" (2-3 bars long) and 7 were "long" (3-4 bars long).



Participants generally rated real examples as real and computer generated examples as computer generated, with a few notable exceptions. Participants generally did not rate at the extremes (1 or 5).



We define inaccuracy as the difference between the real classification of an example and the rating the participant provided squared. Participants tended to be less accurate at rating examples of real performances than computer generated examples.



Participants tended to be slightly better at rating examples that were longer than shorter examples. We feel that if we extended our testing to even longer and shorter durations this relationship would be stronger, and we discuss our response to this in the next section and conclusion.

We also plotted familiarity with the two songs vs. average inaccuracy and familiarity multiplied by musicianship vs. average inaccuracy, but did not find any strong trends. This could be because of the limited number of test participants, the way in which we worded the questions, or they could simply be less related than we assumed them to be.

We hypothesized during the testing that participants would become more accurate/confident in their ratings as the test went along (indeed, this is the reason why we threw out the first datum for each test), but this does not seem to be the case. For a longer test, perhaps this would be more pronounced. We plotted inaccuracy by example, and didn't find any strong trend.

Future Work

Although our system functions, there are many tweaks that could be done to enhance the system overall or improve the quality of the "improvisation" output. We will now discuss some of these tweaks, starting with overall system enhancements and then moving on to internal changes.

Although we aimed to make the process of inputting transcriptions and formatted chord lists as painless as possible, it still requires a good bit of time to transcribe a solo and then write the associated chords. To (partially) automate this process a piece of music scanning software such as Sibelius PhotoScore could be used to simply scan in sheet music and construct a MIDI file from it. Getting the chord symbols would be slightly trickier, as there are many different formats of jazz chord notation, but still possible.

We could also extend the output to include realizations of chords. For testing we had to use outside software to create accompanying tracks for our generated "improvisations", but this could be integrated into the program. MIDI chords could be constructed for each of the chords in the output chord list and a logical rhythm section (drums and bass line) could be created to accompany our "improvisations".

There are a number of internal tweaks that we identified would be helpful, but didn't have the time (or means) to implement:

1. Cross-chord-change Markovs

    When generating output, jumping from ChordPair to ChordPair in our current program has no memory, that is, the beginning of a chord is not influenced by the notes that were played at the end of the previous chord. Adding Markov models that are used solely to transition between chords (modeled after chord transitions in the input data) could be helpful to smooth this transition.

    We identified this, along with numbers 2 and 3 below, as one of the primary reasons why listeners were able to separate "fake" solo fragments from real ones in our testing. We discuss this more in the conclusion.

2. Reinforcing phrasing

    A musical phrase consists of a beginning, middle and end which make sense together, something that our program right now does not take into consideration, so incredibly long phrases (even with long silences being privileged more than anything else) and short, disjoint phrases can be generated. If we were to group the notes in our input into clusters across chords, label them as a particular moment of a phrase ("beginning", "end", etc.) and create "vocabulary" requirements for our output (for example, choose a beginning cluster after we have gotten an ending cluster) this could help with phrasing.

    The Impro-visor system, developed by Robert Keller, does something similar with groups of chords, called "harmonic bricks", and Al Biles' GenJam handles this by grouping notes into "licks" or "chromosomes" that represent a concise, complete musical idea.

3. Rhythmic quantization

    At the beginning of each chord the "improvisation" is exactly in time, as it begins with several notes flush with the beginning of the chord. However, as time goes on, it is possible for the notes to "drift" away from the metrical grid, which creates quite displeasing, easily identifiable as "fake", results. It could be helpful to impose stricter guidelines on the rhythms of the notes we generate, altering the output of our Markov model to fit the context.

4. Restricting pitch contour

Although storing the notes in our Markov model in a generic way allowed us to tap into many more tuples of notes that will be consonant with a given chord, the effect of this choice, having to pitch-shift our notes in relation to the *actual* chords they are associated with in the output, can cause notes in the "improvisation" to be in a different register (pitch space) than they are in the input. We did nothing to alter or restrict output pitch contour. Keller handles pitch contour in Impro-Visor by representing melodic contour as a series of up and down movements, which are associated with a particular "lick". Something similar could be added to TMIG.

5. Dynamic weighting of silences/normal notes/short notes

In our program we tweaked settings for weightings of following notes for different types of notes to be the following, static values:

short notes = 1
long notes = 15
short silences = 30
long silences = 45

These worked well for our specific cases (3 songs of John Coltrane), but to generalize these settings it would be possible to analyze the trend of these different types of notes in the input and adjust these weightings dynamically.

Conclusion

In our testing, since we used order-3 Markov models, any group of four notes (except for three notes that cross a chord change) will be notes that Coltrane actually played over that chord in our input data. Because there are unique Markov models for every pair of chords, everything generated is consonant (because everything Coltrane played was consonant). That is, there are no "bad notes". However, because we did not implement a way to Markov-generate the first note on a chord based on the last note(s) of the previous chord, there is generally no logic/flow in lines when heard in the context of 8 or more notes. The more times the listener hears a phrase "dead-end" and start over as chords rapidly change, the less real or human the line sounds. This is reflected in the data: listeners were more accurate in determining the "fake"/"real" classification of examples that were long (3-4 bars) than in those that were shorter (2-3 bars).

Even if cross-chord-change Markovs had been implemented, it is likely that this trend would have still applied. We suggest several possible ways to enhance the phrasing, melodic contour, and rhythmic precision in our current system in the previous section, but even with these enhancements implemented (as in Impro-Visor and GenJam), the "improvised" output still feels "synthesized" to us. While it is easy to make a computer generate solos that are consonant, the subtlety of a well-crafted phrase is far more complex and intangible. This is analogous to what happens with Markov-generated English. When the Markov process is low-order, we get English-sounding words or small clusters of words that fit together, but to get coherent sentences with higher order generation, we inevitably end up copying full sentences and aren't really generating anything new (note that here we still lose meaning at the paragraph level). Many people are able to play consonant lines over chord changes, but what distinguishes a master such as Coltrane is the ability to make good sentences and even paragraphs with their improvised lines.

APPENDIX

GenJam paper: http://igm.rit.edu/~jabics/GenJam94/Paper.html

Impro-visor: http://www.cs.hmc.edu/~keller/jazz/improvisor/

Jazzerbot: https://code.google.com/p/jazzerbot/

Microsoft Songsmith: http://research.microsoft.com/en-us/um/redmond/projects/songsmith/