Music Generation with Probabilistic Finite Automata

Benjamin Jenkins

CMPT 440 - Formal Languages and Computability

Dr. Pablo Rivas

5/14/2019

Abstract

A piece of music can be represented as a sequence of combinations of notes. These combinations of notes can be thought of as states; each time in the song where a new note or combination of notes begins playing is a new state. In any given song, the same states will appear many times, so a probabilistic finite automaton (PFA) can be constructed based on the sequence of notes, with a transition table that represents the probability of moving from one state to another. A MIDI (musical instrument digital interface) file represents this sequence with messages that describe at which point in time a note is turned on or off. The messages also contain velocity data, which indicates how loud the note is played, and the file may contain several tracks which represent the notes played by different instruments. This project uses a PFA constructed from the information in a MIDI file to generate a new piece of music. For simplicity, the duration of each note in the new piece is constant, velocity data is not considered, and only one track of the MIDI file is used to construct the PFA.

Introduction

Probabilistic finite automata (PFA's) are an extension of non-deterministic finite automata (NFA's). The transition function in a PFA describes the probability of a particular state transition occurring, given the current state of the PFA. The initial state (q_0 in an NFA) is represented by a stochastic vector which gives the probability of the PFA starting in a certain initial state[1].

Music generation is one case where PFA's are particularly useful. The sequence of notes in a song can be viewed as a sequence of states, and a sequence of states can be used to create a PFA. In a song that contains the sequence of notes E, D#, E, D#, E, B, D, C, A, there is a 0.66 chance of transitioning from an E to a D#, and a 0.33 chance of transitioning from an E to a B. Combinations of notes can be treated in the same fashion as single notes. These probabilities are reflected in the transition table of the PFA that is constructed from a song, which will be used to generate a similar song.

A state in a music generation PFA can contain several pieces of information. The most important information is the note or notes that are played when the state is reached. It is also possible to include in the state the duration of each note and the velocity (loudness) of each note. This information can be easily extracted from a MIDI file, which contains one or more tracks, each containing a sequence of messages. Each message indicates that a particular note should be either turned on ("note on") or turned off ("note off"), and also indicates the velocity of the note in the case of a "note on" message. Because the file only contains "note on" and "note off" messages, the duration of the note must be determined by calculating the distance between the on and off messages in MIDI ticks if it is to be included as part of the state. The file may also contain meta-messages, which indicate other information about the song such as the resolution and tempo. The resolution is the minimum number of ticks between each "frame," where "note on" and "note off" messages are found. The tempo is indicated in microseconds per beat.

To construct a PFA from a MIDI file, first, a track is chosen and the resolution is read from the first meta-message containing resolution information. Then, states are constructed based on the messages found at this resolution. If the resolution of a track is 240, the "note on" messages at tick 0 will be used to construct a state, messages at tick 240 will be used to create a state, then messages at tick 480, and so on. In the construction of each state, the velocities are read from the "note on" messages, and note durations are determined by scanning the rest of the file for the "note off" message that corresponds to the note found in the "note on" message and calculating the number of ticks between them. Creating a state from each tick at the track's resolution results in a sequence of states, many of which will be the same. This sequence is used to create a transition table which describes the probability of transitioning from one state to another in the song.

Detailed System Description

In this project, the states are constructed by considering only the notes played at each tick at the resolution of the MIDI file. It is certainly possible to also consider the velocity and duration of each note, but for simplicity and to reduce the number of total unique states (which leads to increased variation in the generated music), I have not done so. My program operates as described in the introduction, a sequence of states is generated from the MIDI file and used to construct a PFA based on the probability of moving from one state to the next. In my implementation, each state in the sequence is inserted into a map, which maps states to a list of states found immediately after that particular state in the sequence. Insertion into the map prevents duplication of states within the map since states are hashed based on the notes contained within the state, which are always kept sorted. To produce the list of states found immediately after a particular state in the map, the sequence of states in the song is traversed, inserting the next state in the sequence into the list associated with the current state.

To generate music from the PFA, my program first selects a random starting state from all available states in the map. A new sequence of states is then generated using the PFA. Each successive state is chosen by selecting a random state from the list of states found after the current state. The lists of states in the map can contain duplicates, so if a particular state is found immediately after the current state more often in the song, there will be a higher probability that that particular state will be selected from the list. Once the new sequence of states is complete, the program assembles a new MIDI file based on the generated sequence of states.

An example of the transition table that would be created from a simple song (figure 1) is supplied in figure 2.

C, C, G, G, A, A, G, F, F, E, E, D, D, C

Figure 1. The first 14 notes of the song "Twinkle Twinkle Little Star," which were used to create the transition table in figure 2

Note	**A**	**C**	**D**	**E**	**F**	**G**
A	0.5	0	0	0	0	0.5
C	0	0.5	0	0	0	0.5
D	0	0	0.5	0	0	0.5
E	0	0	0.5	0.5	0	0
F	0	0	0	0.5	0.5	0
G	0.33	0	0	0	0.33	0.33

Figure 2. A transition table for a PFA created based on the sequence of notes in figure 1. Each cell contains the probability of transitioning from the note in the left column to the note in the top row.

Requirements

To compile this project, the Java Development Kit 8 is required. Running the program requires only the Java Runtime Environment 8.

Literature Survey

Generating music using state machines is not a new concept. Several others have created music generators similar to this one, many capabilities that extend far beyond the limited scope of this project. Others have included velocity and duration considerations in their states, which can enhance the quality of the produced composition, but at the same time might limit the variety of sequences produced. Others yet have included music theory "checks" in their programs, which further enhance the produced songs by ensuring that produced sequences are theoretically sound.

One simple alteration that could be made to this program, which has also been done by others, is to create states based on two notes instead of one. This gives the generator a tendency to group notes together and allows it to give its compositions a better "phrasal structure," as opposed to wandering

through the notes as generators which consider only one note tend to do. While the program created in this project does not do

User Manual

This section describes how to compile and run this project, using the supplied MIDI file (Für Elise) to generate a new composition. You can use your own MIDI files (results will vary) to generate compositions, but you must determine which track of the MIDI file you want to use.

1. Clone the repository:

```
$ git clone https://github.com/benjaminjenkins1/cmpt440jenkins.git
```

2. Navigate to cmpt440jenkins/prj/musicPFA_project:

```
$ cd cmpt440jenkins/prj/musicPFA_project/
```

3. Compile the project with javac:

```
$ javac Driver.java
```

4. Run the Driver program

```
$ java Driver
```

5. When prompted for a file, type the name of the file you wish to generate music based on (furelise.mid is the supplied file). The main track in this file is track 1, so type 1 when prompted for the track if you are using the supplied file. Specify the desired output file name when prompted.

```
Filename: furelise.mid
Track number: 1
Outfile: out.mid
```

6. The generated MIDI file will be placed in the current directory. It can be played with a variety of programs, including an online player at midiplayer.ehubsoft.net.

Conclusion

Although this implementation of a PFA-based music generator does not include features such as duration and velocity considerations, music theory checks, and options to consider the previous two notes in state transitions instead of one, it is sufficient to demonstrate the generation of new compositions that sound similar to a given composition using a PFA. Testing of the program has revealed that it produces the best results when used with pieces that contain notes that contain notes with mostly consistent durations, such as the piece supplied as a sample file (Für Elise). This is to be expected since the program uses a constant note duration. Implementation of variable note duration would be straightforward given the current structure of the program, as would be consideration of two notes in construction of the automaton used for generating compositions.

References

1. Rabin, Michael O. "Probabilistic Automata." Information and Control 6 (1963). Accessed May 15, 2019. doi:10.1016/s0019-9958(63)90290-0.