# An Unlikely Composer

## Algorithmic Music Generation through Markov Chains

Zachary Kordeleski

**Abstract**

Music theory shows there is a mathematical system underlying these compositions, why they work and why they sound pleasant (or, at times, unpleasant). By analyzing this system, we can model the structure of songs in a more precise language and often quantify and explain the composition of the music. We can take this further, applying this analysis to even generate music algorithmically. Using probability distributions, Markov chains, other stochastic processes, and the assistance of a computer, we can algorithmically generate songs with the press of a button.

# Contents

# 1 Introduction

In the same way we now generate environments in video games algorithmically, is it possible to generate music in a similar fashion?

Music, throughout history, has been given form in a manner unique to its many composers. Mozart, when composing his Requiem, spent a large portion near the end of his life painstakingly crafting the many layers of the piece until it only finally took form after his death. This, it seems, is the view of how music is generally composed throughout history. Many hours searching for inspiration and many afterward writing and adjusting the parts for each instrument, toiling until everything is just right.

Yet, with the power of modern computers and some analysis of various compositions, we can develop an algorithm capable of generating unique pieces of music without any human intervention. In seconds, we could theoretically accomplish what takes a composer hours, days or even years. Using music of particular genres, we can analyze the patterns in these compositions to gain an understanding of how the music 'develops' throughout the song. Calculating the various probabilities of various transitions between notes or even physically mapping written compositions, we can quantitatively assign values to certain pitches and probabilistically determine the following note until we have generated an entire song.

In Section 2, we will explore one method by which this can be accomplished.

# 2 Probabilities and Pitches

One of the first ideas that comes to mind when attempting to tackle this problem of composition is to consider each note as a step in a process. We begin with a single note (or chord) and then transition to the next and so on until we reach the end goal or finale of the piece. In this way, each subsequent note depends upon the previous. In other words, we can calculate the probability of transitioning from one pitch to another, even accounting for rests and timing if we so please.

## 2.1 Basic Methods

A simple implementation of this involves nothing more than calculating the probability that a particular note may occur. For instance, let us consider a vector $v$ with one row and $n$ columns:

$$\begin{bmatrix} P_{A1} & P_{B1} & P_{C1} & P_{D1} & ... \end{bmatrix}$$

We could consider each element in this vector $P_{Nj}$ to represent the probability that a particular note $N$ in a particular octave $j$ occurs. As an example,

$$\begin{bmatrix} .1_{A1} & .5_{B1} & .1_{C1} & .3_{D1} \end{bmatrix}$$

This tells us that there is a .1 chance of an A1 occurring, a .5 chance of a B1 occuring, a .1 chance of C1 occuring and a .3 chance of D1 occurring. There are a few things to notice from this. One, the row sum of all the probabilities is 1. This may seem obvious, but it is important that a note always occurs. We could, perhaps, add in a probability that no note occurs (a "rest") but still the probabilities would need to sum to one.

The second thing to note is that our song is currently quite simple and consists of only four possible notes: $A1, B1, C1, D1$. Moreover, we simply filled in this vector using random probabilities. In reality, there are a number of factors that would dictate the probability of certain notes occuring: The key we are playing in, the genre of music, time signatures etc. If we were to simply sample use these probabilities to create a sequence of notes, we might be disappointed with the result. Thus, to better our results, we could sample actual music to calculate the chances that each note occurs. All we need to do is count the number of times a note occurs and divide it by the total number of note occurrances overall. Analyzing the first bit of a simplified "Ode to Joy", for instance, would give us this vector instead:

$$\begin{bmatrix} .200_{C4} & .233..._{D4} & .300_{E4} & .133..._{F4} & .133..._{G4} \end{bmatrix}$$

In total, we sampled 30 notes. We see in this case that E4 is the most popular note, whereas F4 and G4 are rather rare. This would result in us having very few F4s and G4s in the song we generate.

Populating the list using actual music let's us do a few things but, namely, it ensures that we are playing within a particular key. In music, a key is a

set of notes used when playing a song (e.g. C major, D major, D minor etc). If we were playing a song in C major but suddenly played a note NOT in C major (e.g. D sharp), we would hear the dissonance. For our purposes, this dissonance is not favorable as it could recur repeatedly, ruining the 'feeling' of the piece. Let's use these probabilities to generate a short sequences of notes. There's a few ways to do this, but one of the easiest to do by hand is to return back to the list of frequencies write out each note in a list:

$$\begin{bmatrix} C4 & C4 & C4 & C4 & C4 & C4 & D4 & D4 & ... \end{bmatrix}$$

In other words, we've created a list with 6 Cs, 7 Ds, 9 Es, 4 Fs, and 4 Gs (totaling 30 notes) and thus the probability of choosing any particular one from the list is as we calculated. All we do now is use a random number generator between 1 and 30 and select the note in that position as the note to play in our song. Doing this for 8 iterations, we can generate the sequence of notes:

$$\begin{bmatrix} E4 & E4 & C4 & D4 & C4 & E4 & F4 & E4 \end{bmatrix}$$

Which is, in fact, our song! You may be thinking though at this point that this sort of generation could end quite badly, generating very dissonant, choppy music. If you were to think this, you would indeed be correct. Music is more about the chance a note occurs, its about the chance a note occurs in sequence with other notes. This is the driving force behind the Markov Chain approach.

sectionMarkov Chains

As stated previously, a Markov Chain is a tool we can use to take into account probabilities of things occurring in sequence. But before we start using them, let's begin with a definition:

**Definition 1.** *A right stochastic matrix is a square matrix whose entries are real numbers in the interval [0,1] and have a row sum of 1. In other words:*

$$\begin{bmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,n} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,n} \\ p_{n,1} & p_{n,2} & \cdots & p_{n,n} \end{bmatrix}$$

$where p_{1,1} + p_{1,2} + ... + p_{1,n} = p_{2,1} + p_{2,2} + ... + p_{2,n} = ... = 1.$

We have considered here a right stochastic matrix and so all of our multiplication will be done with the vectors on the left side and matrix on the right.

This matrix is also known as a *transition matrix* as it will be responsible for the transitions in our Markov chains. So, how do we use this? Well, first we recognize that each element in the matrix is between 0 and 1 and, moreover, the row sum is 1. This makes it the ideal candidate for elements that are probabilities. Now consider each element as the probability that the note in the row transitions to the note in the column (e.g. $p_{1,1}$ is the probability that an A goes to an A; $p_{1,2}$ is the probability an A goes to a B etc).

Now consider a row vector of n elements that are probabilities a certain note (determined by the column) occurs in the song. If we multiply this vector by our $nxn$ Stochastic Matrix, we will get another $1xn$ vector with new elements and new probabilities of notes occuring. Let's do an example.

**Example 2.1.** *Consider a song with 2 notes: A and B. The following probabilities have been calculated:*

1. $P(A \to A) = .4$
2. $P(A \to B) = .6$
3. $P(B \to B) = .3$
4. $P(B \to A) = .7$

*where $P(a \to b)$ is the probability that note a goes to note b. Assuming you start on note A, calculate the $1st$ probability vector for the system.*

**Solution 2.1.** *To begin, we write these probabilities given into a 2x2 matrix:*

$$\begin{bmatrix} .4 & .6 \\ .3 & .7 \end{bmatrix}$$

*Now we're given the initial condition that we start on A so we have the initial vector:*

$$\begin{bmatrix} 1 & 0 \end{bmatrix}$$

*Performing the multiplication we have:* $\begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} .4 & .6 \\ .3 & .7 \end{bmatrix} = \begin{bmatrix} .4 & .6 \end{bmatrix}$ *This vector then tells us that, if we begin on an A, the following note has a .4 chance to be an A and a .6 chance to be a B.*

You can see how powerful a tool the Markov chain can be when we put it to good use. While it's not a major focus of our goal, it's worth noting a particular phenomenon regarding these chains. Let us now consider a slightly more complicated example:

**Example 2.2.** *Consider the previous state described in the example above. Calculate the second probability state vector.*

**Solution 2.2.** *We've already shown the result above for the first state. The calculate the second, we'll consider the first and move from there. To get the second state vector, we take the result of the first multiplication and multiply again!*

*Performing the multiplication we have:* $\begin{bmatrix}.4 & .6\end{bmatrix}\begin{bmatrix}.4 & .6\\ .3 & .7\end{bmatrix} = \begin{bmatrix}.34 & .66\end{bmatrix}$ *This vector then tells us that, if we begin on an A, the following third note in the song has a .34 chance to be an A and a .66 chance to be a B.*

There are two things to learn from this example:

1. To find the $n^{th}$ vector of a system, we can repeat the process of multiplication.
2. If we continued this multiplication on for a long time, we could potentially arrive at some steady state.

This brings us to the following remark:

**Remark 2.1.** *Given an initial vector A and stochastic matrix T, the $n^{th}$ probability vector $v_n$ of the system can be found through repeated multiplication. This reduces to:*

$$v = (((AT)T)T)T... \rightarrow v = AT^n$$

*And thus the long term, steady state probability vector $v_s$ of the system can be found:*

$$lim_{t\rightarrow\infty} AT^t = v_s$$

Notice that this does not necessarily give us a result but, if there is a steady state probability vector, it can be found this way. Applying this to our music generation, we notice that its possible for our choice of notes to

eventually get "stuck". As we approach a systems steady state vector, the music would start to sound less varied as the probabilities cease to fluctuate. However, this does not necessarily ruin the idea and can, in fact, improve it.

Having established what is necessary to generate our music, we can now move forward to the algorithm.

# 3 The Music Generation Algorithm

The actual implementation of this theory for music generation practically requires the aid of a computer and some knowledge of programming. The algorithm has been completed in a language known as *clojure* and the source code can be found attached as an appendix. Instead of going through the entirety of the code, we will describe the algorithm in general, noting our method and results.

We will break the algorithm down into steps:

1. Parse a midi file to create a sequence of each sound and its duration played in the midi file.

2. Partition the sequence into pairs of n and count the frequency of occurance of each transition.

3. Export these frequencies into a matrix and normalize the matrix by dividing each row by the total number of occurrences in that row. This will be our stochastic matrix.

4. Choose a random starting position and create the initial state vector.

5. Multiply this vector by the stochastic matrix to get the next state vector.

6. Choose a random number $0 < q < 1$. Compare this number to the first element in the state vector. If q is greater than this element, continue onto element two. Sum element one and two and compare again until the element is greater than q.

7. Identify the key associated with the index of this element in the state vector and record it.

8. Repeat for x iterations to have a song of x notes long.

Some example outputs of this algorithm can be found attached. Before we reach our conclusion, it is instructive to go through a few of the steps and explain some of the particulars. Firstly, we must have a sample piece of music before we start the algorithm in step 1. This is our source data and is necessary to 'count' the notes. After this step, we partition the resulting array into n parts. In our cases above, n = 2. We calculated the chance that a single note went to a single note. If n were instead 3 or 4, we would calculate the chance that a sequence of notes went to a single note. This can drastically improve the quality of output and such Markov chains are known as "Higher Order Markov Chains".

At this point we simply do some counting to construct our matrix and choose a random value to start. However, when we pick our random number q to calculate our second note, many people get caught in this step. Remember that the row sum is always 1. We want to be able to go through our list and always guarantee an output lest we break our algorithm. Thus, by summing the tested element's probability and the new element's we guarantee an eventual 100 percent probability of success.

This leaves the last point. After we have exported all the elements to a list, they must be played somehow. In our case, we have used a plugin for clojure known as "Overtone" to play the sequence of notes. It has a handy function known as "midi-¿hz" which translates directly between the midi sound keys and note frequencies to make the playing much easier, but there are many possible methods.

One thing worthy of considering is the possibility of improving our output further using higher order chains. Before we define this, let's try to understand the logic first. Our problem with the very first method we tried in Section 1 was its oversimplification of the process. It only considered the probability of a note occurring, not the probability of a sequence of notes occurring. We improved this by considering the previous note to each and calculating the probability of a given note occurring after another. We could go one step further and consider *two* previous notes instead of one. This gives us the following definition:

**Definition 2.** *A Markov Chain of order n is a Markov Chain which considers n previous states. In other words, the Markov chain has a memory of n states.*

# 4 Conclusion

We've seen how we could use Markov Chains to generate a string of notes. By parsing through a given piece of music, calculating the probabilities of notes occurring, then considering some intitial state, we can take a "walk" down infinitely many paths, each generating a potentially unique sequence of notes. In the above examples, we only considered the probabilities of notes occurring, but this can easily be extended to include note durations as well. By sampling both the note AND its duration, we can determine the likelihood of a given note being of a certain length and, after the note has been chosen, we could insert a step in the algorithm identical to the note choosing process to assign it a duration. Another improvement on this method includes increasing the order of the Markov Chain, as discussed in the final bit of Section 3.

# References

[1] Gustavo Diaz-Jerez *Algorithmic Music: using mathematical models in music composition, 65-76.* 2000: Manhattan School of Music.