Music Generator Applying Markov Chain and Lagrange Interpolation

Yurui Xu *

Department of Computer Science, Vanderbilt University, Nashville, US

* Corresponding author email: yurui.xu@vanderbilt.edu

Abstract. Algorithmic composition is the process that composes music based on algorithms by formulating specific rules that incorporate relevant music arranging theory and assigning these rules to a suitable computer algorithm. One common approach is to use Markov models to generate note sequences. However, the generated notes do not consider the overall chord progression and the rhythmic pattern of the section, which is not effective when dealing with long multi-track music, especially for more structured works, e.g., classical and pop music. Therefore, this paper introduces an improved approach that uses Markov chains for chord generation, and then uses Lagrange Interpolation to generate melodies and accompaniments for multitrack customized instruments using chord progression notes as anchor points. In this study, only 30 common chord progressions and 30 rhythmic progressions were used as the input data to generate a sample music with cat sounds as the main melody, violins as the sub-melody, and pianos as the accompaniment. It eliminates the mentioned problems associated with the adoption of Markov chains for melodic composition. In this case, it allows for the creation of tonal music with an overall structural, multi-track approach, by using chord progressions as a guide to the melody and even the accompaniment, and by controlling the structural characteristics of the music through rhythmic patterns.

Keywords: Markov Chain; Lagrange Interpolation; Music Generation; Nyquist.

1. Introduction

Music is closely related to human life, yet due to the specialized nature of musical knowledge, it takes a lot of time and resources for ordinary people to compose music, so an increasing amount of work is being done on how to compose music with the help of computers. Early data-based probabilistic statistical methods usually used N-gram or Markov models [1]. Later, Bretan et al. proposed a music fragment selection method to combine new music by computing the similarity ranking among music fragments [2]. With the rise of deep learning, deep neural networks were applied in music generation to solve the above problems. Franklin proposed to use recurrent neural networks to represent the possibility of multiple notes at the same moment, thus enabling the generation of more complex musical sequences [3]. In the work of Hadjeres et al., an RNN-based approach to gospel music generation was proposed, where the model generates multiple vocal parts by using Gibbs sampling [4]. In addition, Yang et al. [5] and Morgren et al. [6] used GANs to generate music by taking random noise as input and regenerating the melody. Despite the extensive research on the music generation problem by a large body of machine learning work, many musical factors have not been fully considered. To be specific, chord progressions and rhythmic patterns are widely found in classical and popular music, where chord progressions can guide the direction of the melody and rhythmic patterns can control the structural features of the music.

The Markov chain can be summarized into the following formula $p^* = A^n p$, where p is a vector describing the current state probabilities, p* is the vector describing the next state probabilities, and A is the matrix describing the transition probabilities between current and next state. As the power n gets greater, p* will finally converge to a stable probability distribution for this Markov chain [7]. In the traditional methods of music generation, music generation has been initially performed based on statistical probabilities, such as using Markov chains for note sequence generation. The existing note sequence is used as the observation sequence, the hidden Markov model is used to obtain the hidden states of the current notes, and the probabilities of all the hidden states are obtained through the hidden state transfer probability matrix. Finally, the output matrix is used to determine the notes with the

highest corresponding probabilities to obtain the generated notes. Each note is considered an observed state, and the probability of the next generated note is predicted by learning the transfer probabilities between notes. In this case, the note to be generated is selected. Markov models are better able to model simple musical sequences, however, the model itself performs sequence prediction with the input of the next moment only related to the state of the previous moment and cannot handle long musical sequences. Therefore, Markov models can only deal with single sequences and cannot handle multi-sequence multi-track music [9]. To solve this problem, this study introduces the idea of generating chord progression rather than melody line using Markov chain, then applying Lagrange interpolation to generate specific notes based on chords.

Lagrange interpolating is a method which generate continuous function using known points on the plain [8]. As a trivial example, to obtain a function with certain zero-points on the x-axis, a_i, the method gives the formula for finding the needed y-value for the given x-value as $f(x) = \prod(x - a_i)$. This formula is generalized to arbitrary points (x_i, y_i) within 2-dimension coordinates:

$$P(x) = \sum_{j=1}^{n} P_j(x), \text{ where } P_j(x) = y_j \prod_{k=1, k\neq j}^{n} \frac{x - x_k}{x_j - x_k} \quad (1)$$

The generated function has a degree less than or equal to $n - 1$ where n is the number of points offered at the beginning.

To obtain a continuous curve that passes through certain notes at specific time, the study applies Lagrange interpolating to obtain this function. The time of the first note of each bar is set as X-value. Y-value is randomly selected from a chord tone within the range of C3 to B5 of current bar's chord. These (x, y) points are used to fit the Lagrange interpolation function. Subsequently, the start time of each note is put into the function to get the predicted note, denoted as A. Finally, a chord tone in the bar's chord of this time is randomly selected, and the note closest to the A after ascending or descending the tone n octaves is put into the final pitch list. Classical tonal music has some strict rules. Compared to contemporary music, it leans to unity more than variety. There is some necessary knowledge required in music this research explored.

A chord is a combination of multiple notes sounding at the same time, and a series of chords forming a sequence is called a chord progression. As shown in Fig. 1, each period in the melody has a corresponding chord, and the "F-G-Am-Em" is the chord progression, which recurs throughout the song. The chord progression can influence the mood tone and melodic direction.

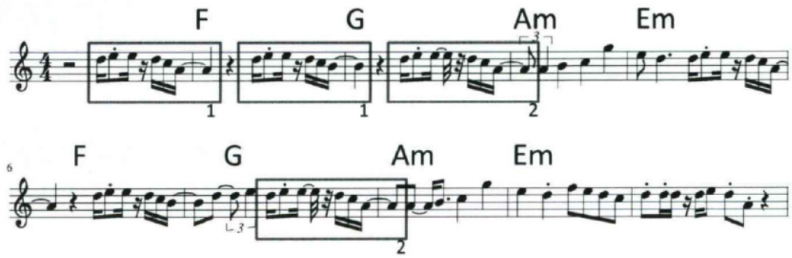

Fig 1. Sample music score.

A rhythmic pattern can be defined as the duration of notes within a phrase. For example, a phrase marked with a box of the same color in the diagram has the same rhythmic pattern, indicating the duration of each note within a phrase. Unlike note-level music generation, classical music is more structured. Much previous work has not considered the structure of songs.

Another characteristic of classical music is orchestration, which means that more instruments are needed to accompany the melody. Classical symphonies usually have multiple tracks, and multi-tracked music requires multiple instruments to work with each other to maintain harmony. There are four tracks in this music: melody, bass, strings, and guitar, where multiple tracks have the root note on note G at moment 1 min 6 sec. This indicates that all tracks remain on the x chord, i.e. they are in harmony with each other. Harmony is a key and fundamental requirement for multi-track music and is one of the conditions for the creation of classical music.

This study applies the following framework to explore classical music compositional rules:

- Harmony and Rhythm are generated using the Markov chain.
- Stepwise motions are realized by using Lagrange interpolation to fit a curve to restrict octaves of notes.
- The piece starts on Chord 1 and ends on chord 1.

2. Methodology

2.1 Nyquist

The chords, beats, and melody generators are written in Python, but are also available in C++. The pitch/melody generator gets its input based on output from chords and beats generators. The orchestration and the final piece output are written in Nyquist. Nyquist is a sound synthesis and sound composition language based on Lisp and SAL syntax. It deals with both signal processing and sound events in a single integrated system [10]. The intention for using Nyquist is that it can directly transfer a string of numbers into scores and then into sounds. Nyquist also implements a series of functions called phasevocoder, which accepts a pitch and a duration as variables, and returns a function that can change the duration and pitch of the original sound input naturally after applying the appropriate envelope. The syntax is *pv-time-pitch(input, stretchfn, pitchfn, dur, fftsize, hopsize, mode)* [10], The second variable is for duration stretching. Here, the study applies an envelope that stretches the waveform in the middle part of any input sound with a stable pitch, so that now the generator allows for adding user-choose sounds as input for instruments.

2.2 Generating Principle

The chord generator will prompt the user to enter the length of chord progression and the starting chord, and generate the subsequent chords afterward. The beat generator is developed based on the chord generator but adds more functionalities with user-defined parameters. The beats generator will prompt the user for 4 values: the number of bars, the BPM, and the time signature (number of beats per bar and which note as one beat). It will first print each note's duration bar by bar, which is used for Nyquist input, and then output a list of absolute start times for each note, and give a list of index for those notes reside on the strong beats. All its functionalities work for all regular time signatures as well as for any BPM.

After generating the chord progression and the beats, the study fits the melody line with a Lagrange interpolation curve and decides what note in a chord to pick based on certain rules. The pitch generator will input a list of chord progression and a list of absolute start times for each note to decide how many pitches need to be generated. It will output a tonic melody line based on the chords given. Finally, the lists of pitches and the lists of duration of notes will be put into Nyquist to add instruments and generate a full score. The flow chart is given in Fig. 2.

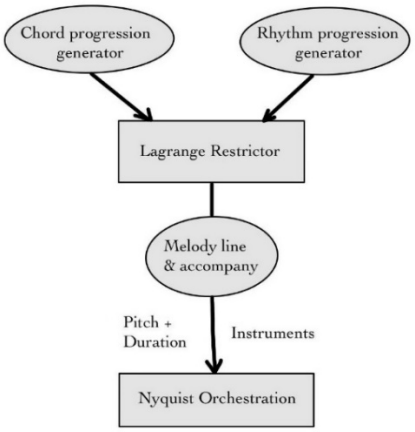

Fig 2. Generating process flow chart.

2.3 Procedure

The dataset input is 30 commonly-used chord progressions. Each progression consists of 4 chords, numbered 1 to 7. Then, one constructs a 7*7 matrix where the columns of the matrix represent the current state and the rows represent the next state. Dividing each column by the sum of the column gives a Markov transition matrix A. Each element represents a possibility. For example, element (6, 4) represents the possibility that chord 4 is followed by chord 6 in all chord transitions starting with chord 4. In classical music, the piece usually starts at Chord 1 (tonic), so a starting state vector of [1, 0, 0, 0, 0, 0, 0] is defined. This is a weighted possibility vector, and all elements add up to 1. Each time the transition matrix A is multiplied with the current p vector to get the next vector. A recursive approach is used to find the final probability distribution $p*$, i.e., $p(n) = Ap(n-1)$. n = 20 is used here, after which $p*$ converges to a stable distribution. The Markov chain of chords and the final probability distribution are exhibited in Fig. 3.

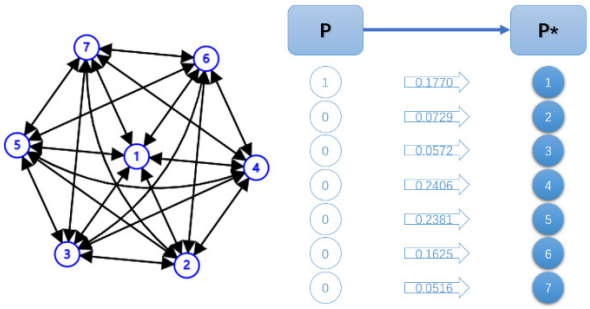

Fig 3. Markov chain of chords.

Subsequently, the program generates a string of number in the range 1 to 7 based on the weighted possibilities $P* = \{P1, P2, P3, P4, P5, P6, P7\}$ for all seven chords, where the total possibilities of getting each chord as the next chord add up to 1. Then the generator picks a random value in the range $[0, 1]$. If the value falls into $(0, p1]$, the result is chord 1, if it falls into $(p1, p2]$, the result is chord 2, etc. Here, $P1 = p1 - 0, P2 = p2 - p1$, etc. Similar to the chord generator, the beat generator calculates a probability distribution for each note value based on Markov Chain and randomly generates rhythm progression for one bar. Afterwards, for every subsequent bar, it applies a probability to determine whether to repeat the last rhythmic pattern or to generate a new one. A 2/3 possibility for getting the same rhythmic pattern of the last bar for the next new bar is applied here for the purpose of testing simplification. It improved the uniformity of the rhythm of generated music, which is an important characteristic emphasized in classical music composition. The generator applies the following representation of notes in order to divide the bar into steps of the smallest note, as illustrated in Fig. 4.

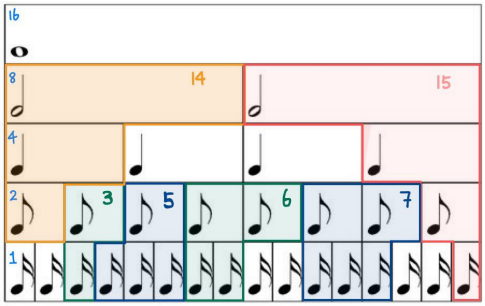

Fig 4. Representation of notes value.

In testing, a sixteenth note is 1 step, thus an eighth note will be 2 steps, etc. This representation has the advantage of easy to grow in the future. For example, simply changing the 32th note to be 1 step will suffice the need for adding tuplets and quadruplets tempos. This gives the formula for

deciding the total number of steps in a bar.

$$\text{total steps in a bar} = \frac{beats}{bar} \times \frac{largest\ note\ value}{note\ value\ per\ beat} \quad (2)$$

Here, the largest note value is 16. Then, assuming 1 full bar has all the notes' relative duration adds up to 1 gives the formula of the relative duration of notes in a bar.

$$\text{relative dur of note in a bar} = \frac{\text{note step value}}{\text{total steps in a bar}} \quad (3)$$

That gives the final formula to calculate the absolute duration of each note using BPM:

$$\text{absolute duration of note} = \text{relative dur of note in a bar} \times \frac{\#beats}{bar} \times \frac{\frac{seconds}{minute}(=60)}{\frac{beats}{minute}(BPM)} \quad (4)$$

For example, for BPM = 60, an eighth note in a 4/4 time signature will have an absolute duration of $0.125 \times 4 = 0.5$ seconds. For every current note, the duration of all previous notes is added up to get the absolute start time. Subsequently, the program selects the first note in each bar, and randomly assign it with a note in its chord $\pm(-1, 2) \times 1\ octave$. Then, one connects these notes using stepwise Lagrange interpolation. For the rest of the notes, this paper picks the note in the corresponding chord that is the closest to the curve. In this way, the final melody fits a smooth curve while still being tonic. After getting notes' pitches and duration, these values are put into Nyquist to combine them together. It is realized using the score generator functions in Nyquist, such as score-gen, score-merge, and make-cycle.

3. Results & Discussion

3.1 Melody and Rhythm Results Analysis

The final product is a set of programs that prompts the user for the length of the music (in how many bars), the starting chord, the tempo, the speed of music (in BPM), and the instruments used as inputs, and a .wav music file as output. As an example, a piece of sample music generated by the program based on the input sample *Torrent* by Chopin is called *Does Electric Chopin Dream of Bionic Cats*. The whole piece generated notes on 120, 60 and 30 bpm as melody lines and accompaniments to create quick and slow tempos contradiction. Fig. 5 shows the melody line of the first four bars between the resultant music and the original piece. It gets the user inputs of 4, 1, 4, 4, 120, and cat, which means 4 bars, chord 1 as starting chord, tempo 4/4, and the instrument of a recorded cat sound respectively.

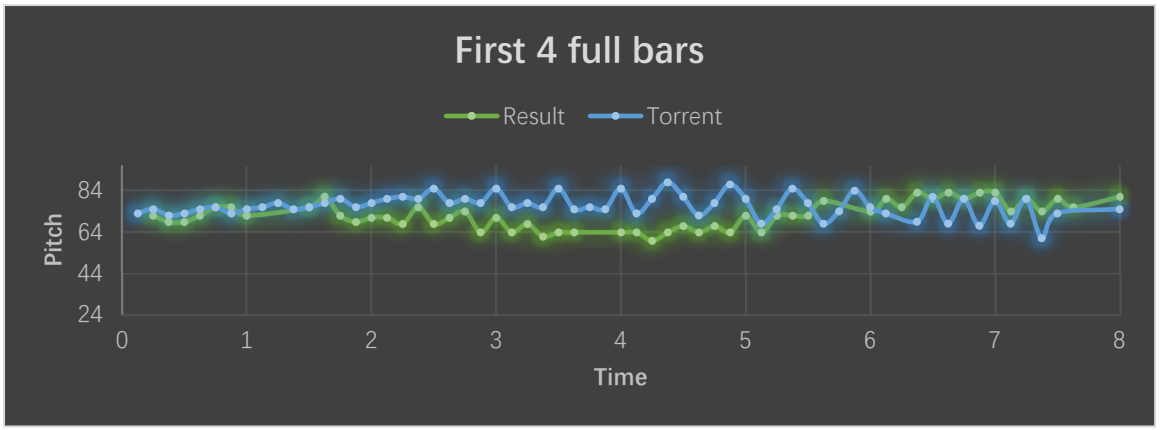

Fig 5. Comparison of result of melody and beats generators with original melody of Torrent.

The standards for analyzing the beats generator include the compactness of notes in a period of time and the occurrence of rhythmic patterns. The original piece has the numbers of notes of 16, 16, 16, 12, while the generated piece has the numbers of notes of 13, 14, 14, 14 for the first four bars. They are not the same, but has the similarity of three repetitions of the same rhythmic pattern and one new rhythmic pattern. Pitch is harder to analyze as there needs to be a balance between originality

and uniformity for music. There needs to be a generally similar pattern of development pitch, but they cannot be exactly the same pitches as the original piece, also due to the fact that they were not in the same tone. The peaks and troughs of the resultant piece are relatively consistent with the original piece. From 0s to 1.5s, they all have a gentle introduction of the opening with little fluctuation of notes. From 1.5s to 3.3s, and from 4s to 7.5s, the generated piece appears a similar pattern of large and regular fluctuations of notes, which loosely corresponds to the pitch pattern of the original piece, but the amplitude of peaks is not the same. In conclusion, the generated piece produces an observable regularity and uniformity emphasized in classical music, but also has certain originality involved.

3.2 Design and Implementation Analysis

This study applies modular design. It encapsulates chord progressions, rhythm progressions, melody generation, and instrument additions into small modules that are designed independently, using only the output values, and finally combining them into a larger musical system. It improves the reusability of these modules. For example, in classical or pop music, multiple instruments create melodies and accompaniment based on the same chord progression in order to ensure harmony, then a chord progression needs to be repeatedly applied to the creation of different melodies, and the generator of the chord progression is split out so that the user can decide the rhythm, tempo, and instrumentation of each small melody based on the obtained chord progression, and the final music obtained is mostly harmonious. These modules can also be directly reused in other projects by simply reassembling and adjusting the interface of the modules according to the actual needs of the product.

In addition, each individual modules have low coupling with independent inputs and single-functional outputs. It makes the program easy to maintain and upgrade, e.g., if the program later wishes to add more rhythm patterns, it only needs to be adjusted for the rhythm generator, without changing other modules, ensuring the high extensibility of each module. Thus, the design and implementation of the programs allow them to be more easily integrated with other projects, can be maintained separately, and managed separately, as well as each single module can be extended and improved by itself without affecting other parts.

3.3 Application Analysis

The set of programs has a size of about 760KB. It did not use any other package except for Lagrange interpolation, so the programs ended up very small and highly controllable. Besides, the sample only includes 30 chord progressions and 20 rhythm progressions, which means the data input is less than 1 KB. The output of the result can be changed by making changes to the small but overtly specific samples. To be specific, if a different tempo is needed, picking another beat sample from different pieces such as largo and presto can generate a totally different piece. The program maximizes the user's freedom in input while also ensures conciseness. It has a high degree of freedom in the value domain as the range of values that can be entered is all positive integers (will ensure that the user enters positive integers). Since the rate of growth of the entire program is linear, it loads very quickly even a large integer is entered. The input method is also free-entered rather than choice-based to maximize creative freedom. It requires only a few numbers to be entered and is very easy to understand and use even for users who do not know much about music.

The most noteworthy thing is the freedom of instruments, which not only means some usual instruments (e.g., piano, violin, saxophone), but also some user-defined instruments. Simply upload the desired sound file, and the pre-designed Nyquist phase vocoder function will automatically generate the rest of the pitches according to the sound and how to stretch or shorten the sound to create different note lengths. For example, the sample music *Does Electric Chopin Dream of Bionic Cats* has the sound of a cat recorded in life as the singer for the melody part.

4. Limitations & Prospects

The research uses branch control to ensure every bar will be a full bar. In reality, there could be notes that traverse two bars. This is a trade-off of keeping uniformity and freedom. In the future, some other probability distribution to see how often a note will traverse will be applied in the generators to add more variety. In each bar, the program also checks if any generated notes hit the strong beats time using if-else branches that group regular time signatures with their strong beat pattern. In the future, the applications will be extended to irregular beat patterns, such as complex (e.g., 5/4 or 7/8), additive (e.g., 3+2+3/8), or even irrational meters (e.g., 3/10 or 5/24). The final product only produces melody in chord tones. But there should be passing tones in post baroque era. Some attempts have been made to achieve stepwise motion, such as setting the strong beats to the tonic, connecting them with the curve and finding the rest notes on the curve, and then merging them with the closest piano pitch. The result turns out to be atonic. The reason is probably that the notes on the strong beats are too short since beats sample are taken from *Torrent*, so there are more nonharmonic tones. In future there will be implementations on chord inversions and counterpoints to restrict them.

5. Conclusion

In conclusion, this paper investigates the grammar-based composition of computer music based on Markov Chain and Lagrange Interpolation. Specifically, with 30 data points as input, the resulting piece is a tonal polyphony with a rhythmic and melodic pattern that is consistent with some laws of the classical composition according to the analysis. Besides, the program is modular, with chords, rhythms, melodies, and orchestrations designed separately to ensure functional independence. In this case, the program is more extensible and maintainable, and new rules can be easily added to improve the freedom of composition and conform to the laws of music composition. In addition, this study uses a simple and free user input format for user interaction to allow more people with less knowledge of music to have the opportunity to compose music, allowing the user to customize the length, starting chord, beat number, tempo, and instrument timbre of the piece with only a few positive integers. Nevertheless, the research still has limitations on different tempos and traversal pitches. In the future, the program expects to add traverse notes between bars, passing tones, as well as irregular beat patterns for the generators to produce music with more variety and closer to the composition of real human beings. Overall, these results offer a guideline for improving the Markov chain's application as well as the overall structural consideration of music pieces in computer music generation.

References

[1] McCormack Jon. Grammar based music composition. Complex systems, 1996, 96: 321-336.

[2] Healey, Jennifer, Rosalind Picard, and Frank Dabek. A new a ect-perceiving interface and its application to personalized music selection. Proc. Wkshp on Perceptual User Interfaces,(San Francisco), 1998. Franklin, Judy A. "Recurrent neural networks for music computation." INFORMS Journal on Computing 18.3 (2006): 321-338.

[3] Franklin J A. Recurrent neural networks for music computation. INFORMS Journal on Computing, 2006, 18(3): 321-338.

[4] Hadjeres Gaëtan, and Frank Nielsen. Interactive music generation with positional constraints using anticipation-rnns. arXiv preprint arXiv:1709.06404, 2017.

[5] Yang Li-Chia, Chou Szu-Yu, and Yang Yi-Hsuan. MidiNet: A convolutional generative adversarial network for symbolic-domain music generation. arXiv preprint arXiv:1703.10847, 2017.

[6] Moore Jack Murdoch, Débora Cristina Corrêa, and Michael Small. Is Bach's brain a Markov chain? Recurrence quantification to assess Markov order for short, symbolic, musical compositions. Chaos: An interdisciplinary journal of nonlinear science 2018, 28.8: 085715.

[7] Verbeurgt Karsten, Michael Dinolfo, and Mikhail Fayer. Extracting patterns in music for composition via markov chains. International conference on industrial, engineering and other applications of applied intelligent systems. Springer, Berlin, Heidelberg, 2004.

[8] Sauer Thomas, and Yuan Xu. On multivariate Lagrange interpolation. Mathematics of computation 1995, 64.211: 1147-1170.

[9] Simoni Mary, and Roger B. Dannenberg. Algorithmic Composition: A Guide to Composing Music with Nyquist. University of Michigan Press, 2013.

[10] Dannenberg R. Nyquist reference manual, 2008.