“John Chain” - Creating Jazz Improvisations with Artificial Intelligence

Ela Fallik, Liron Sade, Yoni Leibner, Shahaf Bassan

Abstract

A music improvisation is the creative activity of immediate composing. In this project, we tried to simulate the process of creating a Jazz improvisation with artificial intelligence tools. We generated a new melody[1] based on pre-existing melodies, using two main stages. The first is choosing a rhythm for each set of musical bars applying different specifically developed **heuristics**. The second is populating the rhythm with musical notes via a **Markov model**. We found this to be an appropriate approach as improvisations are muchly based on the “feel” of the player who many times bases the part that he plays on the previous parts in a “free” manner. We evaluated our model using volunteer questioning and found the order of the Markov chain to be critical parameters. We tested our results with a Turing test and asked people with different musical backgrounds to distinguish between pre-existing melodies and the ones we generated and rate them on how “human” they sound to them. The surprising results gave us a distribution close to uniform, meaning the audience, including musicians, had difficulties in recognizing the real melodies from the generated ones.

Introduction

Previous Works

Previous works were done in the field of generating music pieces. One of the most well known works was done by Prof. David Cope from the University of California, Santa Cruz. Cope, a computer scientist and a composer, created “EMI”[2] (experiments in musical intelligence- 1992) which used a dataset of Bach compositions, and using familiar patterns built an algorithm that generates new similar Bach pieces. He gave the generated pieces to experts of Bach who had a hard time figuring out what were the real Bach pieces. Cope’s way of generating melodies was by breaking the original melodies into small pieces (deconstruction), recognizing common signatures of style and recombining them into new musical pieces. More and more researchers are trying to attack this subject from different angles- some of them include modern approaches of machine learning and artificial intelligence, like the use of neural networks.

[1] A list of musical conventions used in the paper is found in Appendix 1
[2] Cope, David. "Computer modeling of musical intelligence in EMI." *Computer Music Journal* 16.2 (1992): 69-83.

Motivation and Overview

We decided to take on a similar challenge of generating a music composition. We were specifically fascinated by the concept of trying to produce a **Jazz improvisation**. A music improvisation is the creative activity of immediate composing. An improvisation, in contrast to planned composing, is often associated with emotions and spontaneity, meaning the “feel” that the player has while creating the composition. This is what made us so interested in this topic - will it be possible for an algorithm to simulate such an assumingly emotional act such as a Jazz improvisation?

When we first sat to discuss the problem, and after reading previous works that were done in this area we all agreed that the best way to approach the issue using AI is with learning, as it will provide the audience with familiar musical melodies they have heard from other pieces. We knew that the problem of generating a jazz improvisation will include two main parts: generating **rhythms**, and generating **notes**.

While we were familiar with previous works done in the field like the “Emi” algorithm that was done by Professor David Cope, we weren’t sure that was the way to go in our case. We found the work that he did to be very different from our problem, as learning Bach pieces is much based on structured patterns. We, on the other hand, wanted a more “free” approach that will fit the problem of producing a Jazz improvisation. We chose to use the approach of learning via estimating an unknown Markov chain transition matrix from our samples (known jazz melodies). An improvisation is muchly based on the “feel” of the player who many times bases the part that he plays on the previous part in a “loose” feeling manner. The idea was to take a big data set of jazz improvisations for the program to learn from and let it decide what notes and rhythms to play based on previous notes and rhythms.

Although we did know that there is a musical connection between the notes and rhythms played, we decided, due to efficiency issues, to learn each of them separately. This was a decision taken considering that we rather learn the notes and rhythms more in depth (especially the notes), than to have a shallower learning method that includes both of them together.

This approach of using a markov chain may have superfluous consequences:

1. The melody can become too “loose” and with no clear musical structure, moving from one musical phrase to another randomly.
2. Overfitting the data to some of the melodies in the data set.
3. Repeating certain parts of the melody too many times.

These were the techniques we decided to use for dealing with these mentioned issues:

1. Regarding the first issue, we decided to use **heuristics**. Heuristics is a concept more commonly used in search problems. We decided that using it in our case could help us bring some general musical concepts to the melody, that will prevent it from becoming too obscure.

2. Regarding the second issue, we decided to use **stochasticity** during the learning process. In this way we have low probability of repeating the exact results from the melodies we take as our data.
3. Regarding the third issue, we decided to use **exploration** when producing the notes. Again, exploration is not necessarily a concept used in Markov chains, but we found it may be helpful in preventing the notes from repeating themselves too many times, and finding new, original and interesting musical areas to explore.

We were ready to approach the problem using stochastic Markov chain learning, heuristics, and exploration for producing new notes and rhythms for our improvisation.

Data

Our data contains about 100 melodies, all of them in the MIDI format.
A MIDI file is not an audio recording. Rather, it is a set of instructions – for example, which note should be played and for how long. A MIDI file can contain up to 16 channels, the first channel is used to set general settings, as tempo/BPM[3] for the tune, and in the other channels the format is usually one channel for each instrument playing. There are 128 possible notes. We used a python library called mido to process these files. A typical line in these files looks like this: **Message('note_on', note=64, velocity=64, time=32)**
It says that note 64 should be played for 32 ticks of time at strength 64.

Our preprocessing process included the following phases:

1. Classifying the song to one of two groups, “jazz” or “funk”:
   We wanted to separate the music learned in to two different genres of jazz, so that we can hear the difference in the music we get. The first group we took has a classic-jazz sound, using pieces from different artists. For the second group, we wanted to get a more “funky” sound, so we took pieces relevant to jazz-funk artists[4].
2. Adjusting the notes of the melodies to the appropriate scale:
   We wanted all the pieces learned to be on the same scale so we shifted all Midi files, depending on the scale of the melody. We scaled all melodies to be in C Major, or A Minor which is the parallel key[5].
3. Dividing each MIDI file to separate files, each containing one channel:
   Usually, each channel in the Midi file contains a specific instrument. We divided the multi-channel Midi file into multiple Midi files, each containing a single channel. We used only the channels that we deemed relevant to the learning process, for instance the main melody, the baseline and the trumpet solo. It is worth mentioning that we skipped channels that had harmony, as we didn’t want to confuse the logic of learning note process. We also ignored channels that had drums or other irrelevant instruments.

[3] **Tempo** - the speed of the tune, **BPM-** Beats per minute
[4] The full list of artists and melodies is found in Appendix 2
[5] Elaborated under- “Scale transformation used in the project”, Appendix 1

Approach

Learning

Notes

How would we go about generating the notes? We had 128 notes to pick from in the midi format. We decided to model the problem as a Markov chain as described above.
A finite, first order, discrete time Markov chain is a sequence of random variables X1, X2, ... over a finite set of states S s.t. for all n, $P(X_{n+1} = x | X_1 = x_1, ..., X_n = x_n) = P(X_{n+1} = x | X_n = x_n)$.
A finite, discrete time markov chain with memory k will satisfy the property $P(X_{n+1} = x | X_1 = x_1, ..., X_n = x_n) = P(X_{n+1} = x | X_{n-k} = x_{n-k}, ..., X_n = x_n)$.
For a k-ordered Markov chain, each state is the k last notes. The transition matrix describes the probability to transition from the state describing the k last notes to a new note. So for a constant reward, the k-ordered matrix P is defined as $P[(a_1, ..., a_k)][b]$ = p', when p' is the number of times note b was performed after seeing the state $(a_1, ..., a_k)$ in the data. After normalization, P defines a distribution for the notes that should be played next. That is, $P[(a_1, ..., a_k)][b] = P(X_{k+1} = b | X_1 = a_1, ..., X_n = a_k)$
The reward on the states can be manipulated to give a higher value for some notes, like the pentatonic scale (see Heuristics section).

In the beginning, we went just two notes back. We built a matrix of size $128^2 * 128$ with a constant reward value, and for each set of two notes decided what was the highest probability of moving to the next note, based on the dataset we gave it. In such an early stage, we got significant results:

Part 1 audio - the first melody

Nonetheless, we looked for ways to improve them more.

First, when a pair of notes was missing from the matrix (all probabilities from it are zero, meaning no transition exists), we returned a random (0-127) note. Because we started with a small dataset of a couple of melodies, it was a rather common occurrence, so we wanted to improve this method. We also wanted to use higher order of Markov chains, so we created a generic method: given order k, create k matrices, each matrix of size $128^i * 128$, where i=k,...,1. Given last k notes, check if they exist at the k-ordered matrix. If they do, return a note with the appropriate probability, meaning that if $(a_1, ..., a_k)$ were played, for all $b \in \{0, ..., 127\}$, in probability $P[(a_1, ..., a_k)][b]$, return b as the next note. If the k notes were not seen before,

search for the k-1 last notes in the k-1 order matrix. This algorithm proceeds until the last note is not in the first order matrix, (very unlikely) and if so, returns a “close”[6] note.
There was a notable improvement in sound after using this method, as we were no longer getting random notes. However, due to memory limitations, we were capped at order 3.

In order to combat the memory limitation, we decided to shrink the range of notes. Moreover, we noticed that melodies do not use the whole range of notes (0 to 127), but rather concentrate in the middle. We thought that those notes are not necessary, as even a grand piano (which has much more notes than most standard instruments) only has 88 notes, and the 128 notes only harmed the sound when reaching the edges. We decided to shrink our matrix to the 34-98 range while throwing away any data that is outside of the range. This gave us a great improvement, as our melodies were steering away from the stingy notes. Being able to use a higher order markov chain enabled our melodies to sound more complex. We could now generate order 4 pieces.

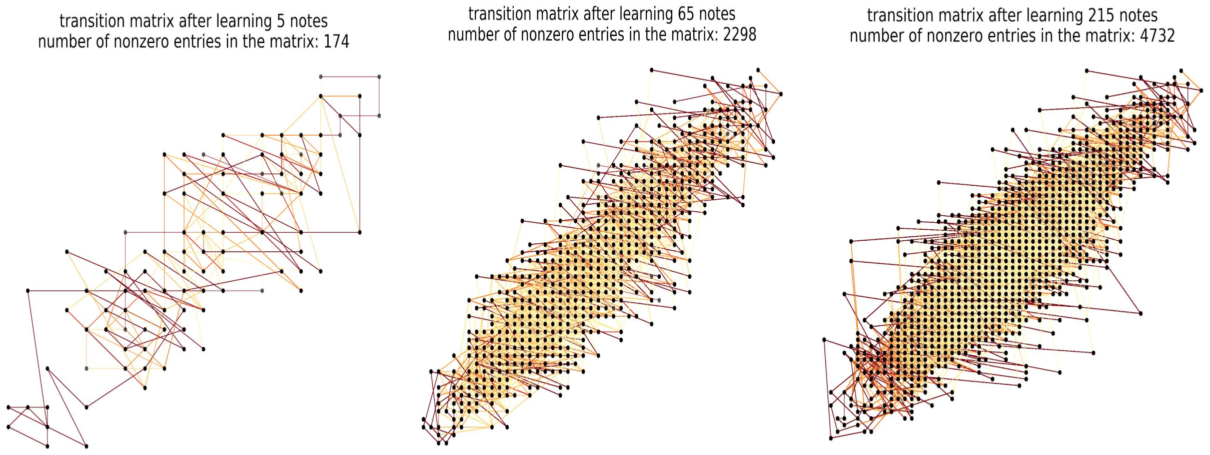

transition matrix after learning 5 notes
number of nonzero entries in the matrix: 174
transition matrix after learning 65 notes
number of nonzero entries in the matrix: 2298
transition matrix after learning 215 notes
number of nonzero entries in the matrix: 4732

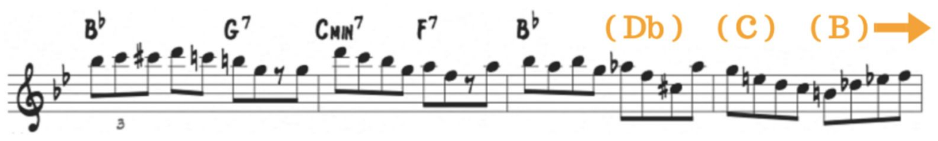
Figure 1. The transition matrix at different stages of the learning. We can see how the transitions concentrate on the main diagonal as more songs are learned.

Part 2 audio - going three back on smaller scale

[6] Described later as “Random Close note exploration”

Rhythms

This part is about a failed attempt to learn rhythms with Markov Chains, that we eventually didn't use. We first approached the problem of creating rhythms in a way similar to the way we learned notes. From the beginning, we knew that the behaviour of rhythms is much more anarchich than that of the notes. When learning notes it is not difficult to normalize all of the data to the same scale. With rhythms, on the other hand, the problem is very complex- one must consider tempo, and metre. This problem is extremely difficult in improvisations were the rhythmic behaviour is even more chaotic. For that reason, we decided to use agreeable Jazz rhythmic patterns. We searched the web for common Jazz rhythmic patterns used in improvisations. We used dozens of different Jazz rhythms, and specifically took rhythmic patterns that were found in the pieces that we learned.



Example for a John Coltrane Rhythm pattern used

We tried to learn the rhythms similarly to the notes, with a Markov chain: For each music bar in each melody, we found the rhythmic pattern that is closest to it (calculated with Manhattan distance) from the pre-selected rhythms.We then updated a transition matrix and according to it we chose the next best rhythmic pattern to move to.
When we finished implementing this approach we didn't find any reason in the way the program moved from one rhythmic pattern to another. It sounded entirely chaotic. When going through the raw data we indeed saw how difficult it can be to learn the exact rhythmic pattern. For example, a bar can have 1 note or 90 notes in the same piece. This was very hard to simulate.

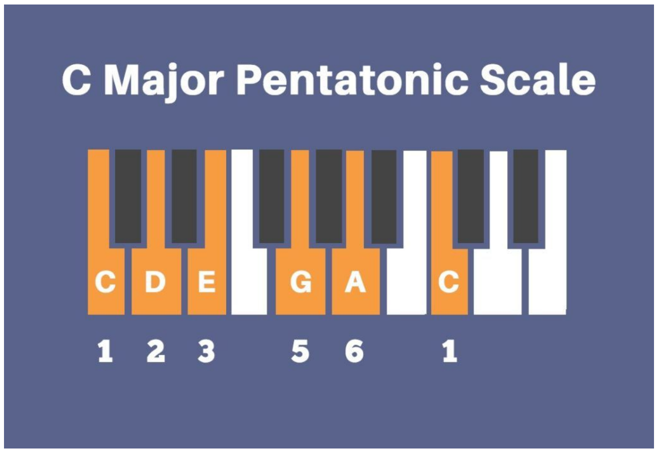
We were working on heuristics for the rhythms as well, and we found them to be extremely efficient. We finally decided to generate the rhythms only based on the heuristics and not on the Markov chain. So the final rhythm of the melody is based on moving between dozens of different jazz rhythmic patterns that we gave it, some are based on the learned pieces and some on purely common jazz patterns. The algorithm decides what rhythm to play based on the different heuristics we built for it.

Heuristics

Heuristics of Note generation:

Pentatonic Scale Heuristic:

After we gave the program large amounts of data to learn from we noticed that the piece indeed sounds much more complex and fluent. However, as the piece became more complex, the improvisation stepped out of the main scale (the tonic scale) too many times. It was becoming hard to follow where the improvisation exactly is. We wanted to constrain the improvisation to play more on the tonic scale, on which it is improvising. We didn't, however, want to interfere too much with the data that was learned. What we decided to do is to take the most tonal notes on the C scale (which is also known as the pentatonic scale[7]), and to multiply the probabilities of those notes in the transition matrix with some factor.



We built this procedure in the following generic way:

$$\overline{P} = P \cdot (x \cdot 1_{group\,a} + y \cdot 1_{group\,b} + z \cdot 1_{group\,c})$$

We chose group a = the C note (the most tonal note of the scale), group b = G and E (the chord notes of the tonic chord of C), and group c = D and A (the two other, less tonal notes of the C pentatonic major scale). We found this technique to be extremely powerful, and even found the piece to be "too tonal", thus the need to evaluate the potential values of X, Y and Z. Potentially, one can use this generic heuristic of the algorithm in order to amplify different types of scales in the improvisation, not necessarily the pentatonic scale, by choosing different groups and multiplication factors. We chose to use it only for the pentatonic scale because we only tried to make the melody sound a bit more "on-scale", and not to interfere with the improvisations of the data that was learned.

[7] Pentatonic scales were developed independently by many ancient civilizations—an indication that pentatonic scales are based upon a naturally occurring phenomenon. A nice way to see the power of the pentatonic scale can be seen in a Ted performance given by Bobby mcferrin in: https://www.youtube.com/watch?v=ne6tB2KiZuk

Note-Narrative Heuristic:
The idea for the use of this heuristic actually started as an accident. On the first days of working on the algorithm we had the program returning to its starting point every 30 notes in order for it not to get stuck. We found that we liked the sound of this, as it made the melody sound repetitive and very easy to follow. We decided to make it into a generic function, giving the program a few notes as parameters for optional starting points. These notes are the narratives that the user can choose to give the program. For instance, a user that wants his piece to be mainly of higher or lower notes can choose appropriate notes that meet his choice. Every X notes, the program chose to randomly place itself on one of these starting points. As we gave the program more data to learn from we found this heuristic to be less important. We eventually did not use this developed heuristic. More importantly, this also gave us an idea to use a similar concept during the learning of rhythms, where the heuristic came to be significantly useful[8].

Heuristics of Rhythm generation:

Fluent progress heuristic:
This is the main heuristic that we finally used for learning rhythms. The general idea for using such a heuristic, came from trying to do something similar to **Gradient descent**. We wanted to compare the similarity between 2 rhythms. This is similar to trying to walk, in highest probability, to the least different slope from your current position (like done in Gradient Descent). So we created two options we found helpful and opened the opportunity for the user to insert his function instead.

$$nextRhythm = \min_{r \in R} F(r)$$

$$F_1(r) = \|lastRhythm - reverse(r)\|_1$$

$$F_2(r) = \|lastRhythm - r\|_1$$

F_1 tries to smooth the transition between rhythms. It is supposed to minimize the difference between the end of one bar, from its continuation, that is the next bar. F_2 on the other hand tries to repeat a more "close by hearing" rhythm, minimizing the difference between the bar and its next bar as is. The L_1 norm caps the tuples so they're in the same length. We ended up using F_2 as it seemed to get better results.

Settled progress heuristic:
Although we found the first heuristic to be very helpful in making the rhythm more fluent, we still found the rhythms to be changing too many times. We decided to build another heuristic method that gives probability X for staying in the same rhythm. We changed X until we found the best fit for the sound we looked for.

[8] More on this in the Rhythm-Narrative Heuristic section

Rhythm-Narrative Heuristic:
As we mentioned earlier, the idea for the use of this heuristic came from using a similar approach with the notes. We added one main rhythm that is repeated with some probability. With Rhythms, we found this approach to be more effective. This was built in a generic way, so a user can potentially choose to decide the rhythm he wants as one dominant rhythm.

Part 3 audio - rhythms example

Exploration

Notes Exploration:
As we did not want our melody to be stuck on playing the same notes and to increase feeling of improvisation, we decided to introduce forms of exploration to our algorithm. We played with two different forms of exploration in this domain.

1. Uniform Distribution
2. Random Close note

Uniform Distribution exploration:
Uniform Distribution exploration means that at probability p1, while looking at the matrices for the next note, we will give all the possible notes (with a positive transition probability) the same probability (uniform). Moreover, if the matrix has only one possible next note, it will ignore it and continue to the next matrix. We decided that the right approach would be to give this kind of exploration a rather high probability so we could hear notes that were learned from the data, but were perhaps less commonly used.

Part 4 audio - high uniform distribution

Random Close note exploration:
Close note exploration means that at probability p2, instead of looking at the matrices for the next note, we will pick a random appropriate close note. That means we will pick one of the following operations: [+12,+ 7, +4, -5, -8, -12] and apply it to the last note played. These are common interval jumps inside the scale in music. As we added more data, the notes concentrated more in the same area, so we decided to introduce this kind of exploration as a way to shift the melody to a different direction leaving inner loops of the matrix and exploring new areas. We decided that the right approach would be to give this kind of exploration a rather low probability as it has a rather random sound.

Part 5 audio - high random close note exploration

Additionally, as mentioned before, the note-narrative heuristic forces the next notes to be predefined ones, at probability p3. This means that the probability of generating a note using the original transition matrices distributions is 1 - p1 - p2 - p3. The close-random exploration and note-narrative heuristic can cause a jump in the melody. We didn’t want these constraints to have a “random jump” effect on the rhythm of the melody. That is why we defined two new probabilities p’1 and p’3 for note-narrative and close-random exploration within a bar, while p1 and p3 are the probabilities for the beginning of a bar. We gave higher probabilities for such an occurrence to happen at the beginning of the bar. When these jumps happen at the beginning of a “musical sentence” they interfere less with the flow of the melody.

Harmonic musical extensions

We wanted to add harmonic effects to our final generated melodies. We added three main extensions:

1. The option of another channel playing the exact same melody in an octave interval from the original melody.
2. The option of another channel with a different generated melody playing parallel to the original melody.
3. Augmented seventh chords[9] played up from the current note of the piece played every random interval. These are chosen only from notes that are on scale.

The generated piece alternates between these three options according to the parameters.

[9] Explained on Appendix 1

Results

Model Assessment

While working on this problem, we found that we created a complex model, which has many methods and heuristics and takes in many different parameters. We wanted to assess the model more clearly and compare the results with different parameters . We chose the parameters that we saw as the most significant: the size of the training set (number of melodies being learned), and the order of the markov chain. We took a group of volunteers and had them listen and rate short melodies while changing each one of the parameters.

<u>Size of training set:</u>
We estimated that the size of the training set (the number of melodies we give the program) is one of the most dominant parameters of our model, and we wanted to assess it. This estimation came from observing the large effects of adding data on the transition matrix. We zeroed all other parameters of the model - no exploration and no heuristics. We gave the notes a constant rhythms and produced 30 second melodies that use random training sets of melodies from our dataset in sizes of: 15, 30, 45, 60, and 75, and let a group of volunteers rate them. Surprisingly, as shown in graph x, we found no significant effect in the change of training set size.

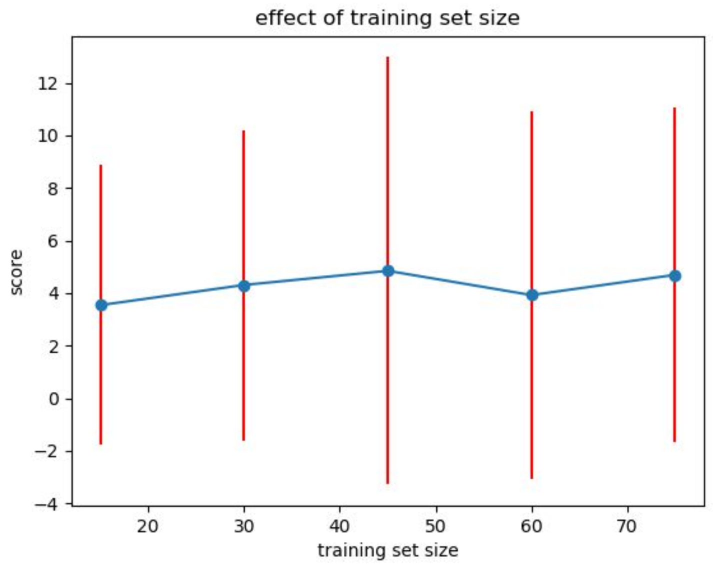


Figure 2: The effect of the training set size (number of melodies) on the score given to the produced melody.

Order of the Markov Chain:

We wanted to find the influence that the order of the Markov chain during the process of learning notes has on the generated melody. We estimated that the general method of going back on the chain will make the piece sound “closer” to real melodies, however, going back too much can overfit the data. We had volunteers evaluating 30 second pieces, learning from a constant data set of 30 melodies, for orders: 1, 2, 3, and 4 (above that was impossible with our computing power).

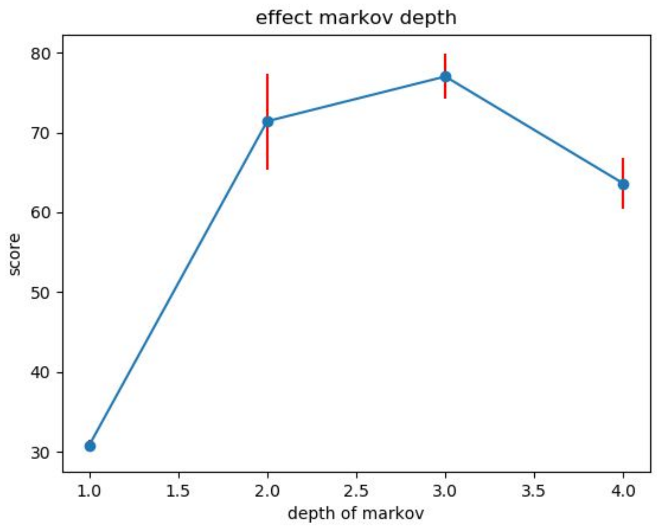


Figure 3: The effects of the order of the markov chain on the score given to the produced melody.

We found these results to be convincing. Clearly, when the order of the markov chain is 1, the sound is significantly lower. Moreover, order 3 might be the “overfit roof” for our model and data, although it is hard to say for sure as the variance around 2, 3 and 4 is quite high. Nevertheless, above order 2 we found similar results. It is worth to mention that this graph can change for our model using different data.

The final Melodies

We ran the program with all the different tunes as the data set. We uploaded the results to a channel on youtube that can be found on: **John Chain - Chaining Bells**

We also wanted to test the final melodies, under the use of different parameters. As mentioned previously, we separated the data we had into two groups of melodies - one containing classic-Jazz melodies and another containing funky-jazz melodies.

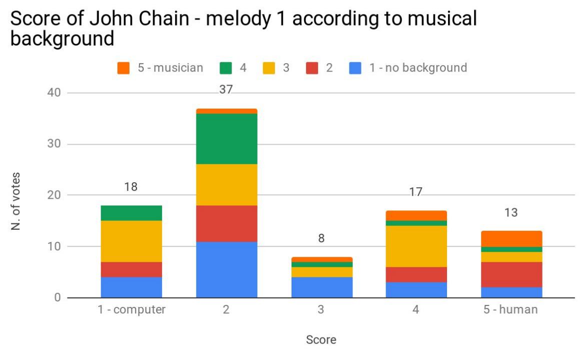
John Chain - Funk

John Chain - Jazz

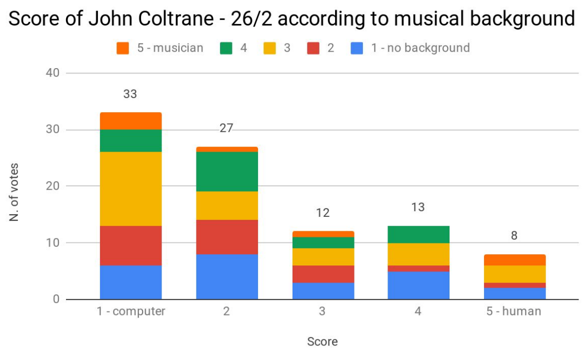
The Turing Test

We wanted to evaluate the success rate of our work as opposed to real jazz improvisations. We chose to do so using a Turing test. The turing test is notoriously known as a bad way to estimate human intelligence. Nevertheless, we believe that in our case the test was the best way to go, as music, and art in general is subjective and the beauty of it is in the eye of the beholder.

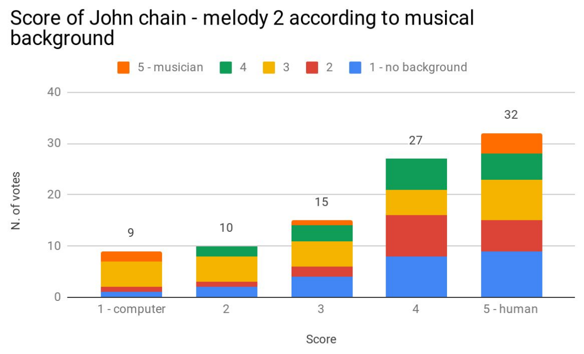
We took 20 seconds sequences from three melodies that our program generated, and three real melodies from well known jazz musicians, and had people listening to them in a youtube video, and trying to distinguish between the pre-existing melodies and the ones our algorithm generated[10]. We had them rate all six melodies on how “human” they sound to them on a scale from one to five. A total of 93 people participated in the test. These were the results:



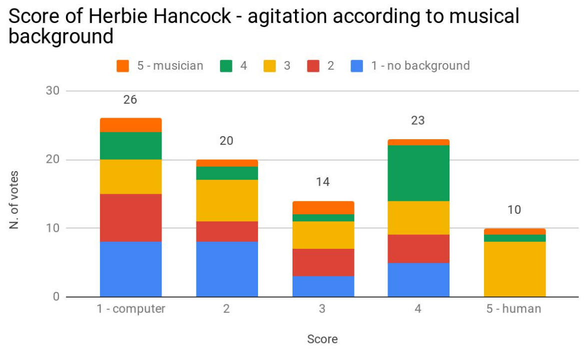
John Chain - melody 1



John Coltrane - 26/2

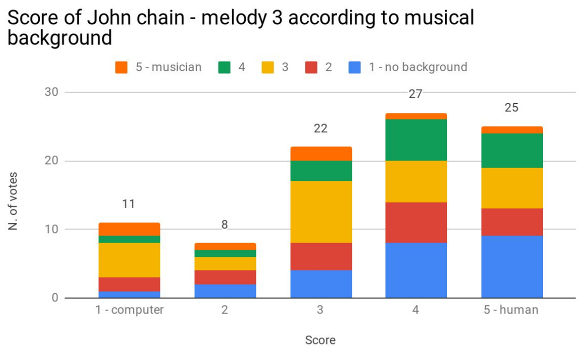


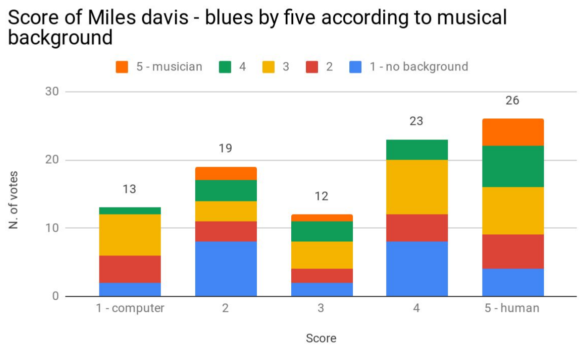
John Chain - melody 2



Herbie Hancock - agitation

[10] The Turing test can be found on: https://docs.google.com/forms/d/e/1FAIpQLSeRpL-GDA0brPtQ2huu-XEK8GqWfJE1L0C7CJaXyzR3bbNq2g/viewform





John Chain - melody 3 Miles Davis - blues by five
Figure 5: The results of the Turing test, for each melody, according to musical background.

The results clearly show that the audience had a hard time finding what are the short human-composed melodies, and what are the short melodies produced by our algorithm “John chain”. The first melody produced by the program had a high rate of votes for sounding like a computer. However, two sequences from human melodies (John coltrane - 26/2, and Herbie Hancock - agitation) also had high votes of sounding similar to a computer. The most “human” sounding melodies were found to be the second and third produced (“John chain”) melodies, and the sequence taken out of Miles davis - blues by five. These following scores can be seen in a graph showing the average score for the six melodies:

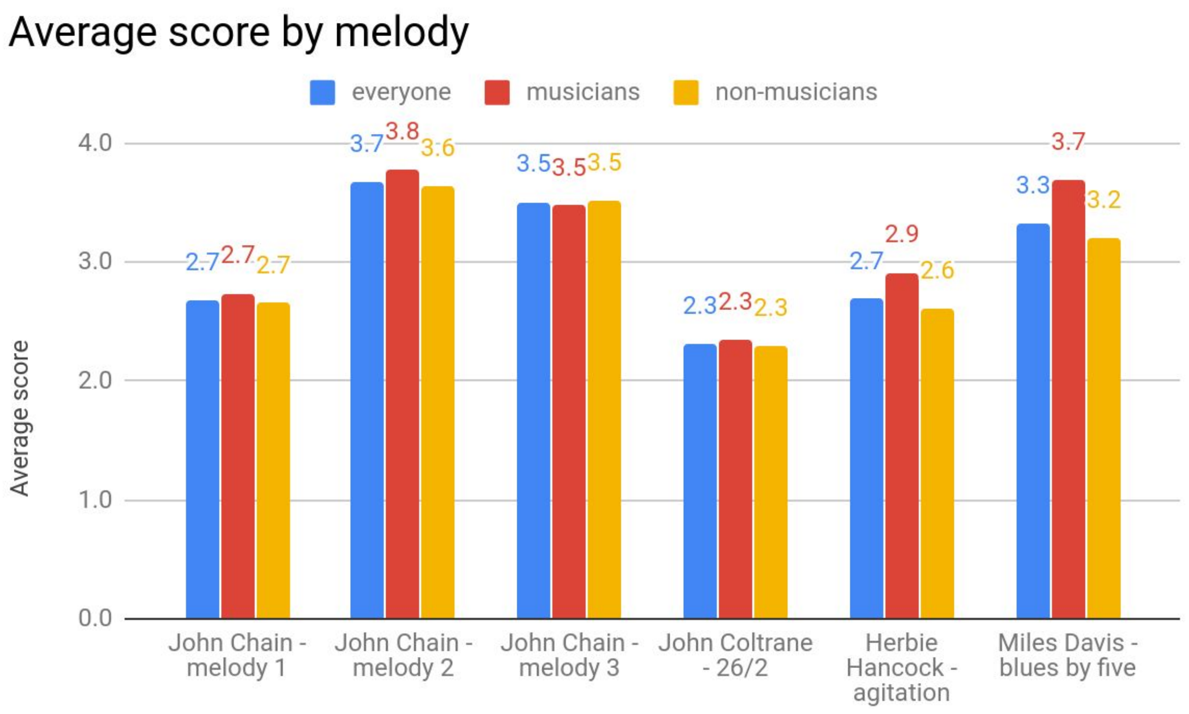


Figure 6: The average results of the Turing test, for each melody, according to musical background. Musicians are people that graded themselves as 4-5 on knowledge in music, and non-musicians are all the rest.

We looked at the same results for participants in groups 4 and 5 of highest musical background (a total of 21), and the non-musicians (a total of 73). The results showed a close to uniform distribution, with similar outcomes. The main difference was that the musicians did slightly better in recognizing the human melodies as human. However they did about the same as non-musicians in recognizing the generated computer melodies as a computer. The musicians group also generally had a higher human rating average, meaning that they were altogether more likely to rate any one of the melodies as human.

Conclusion

This project started off with a quite simple Markov - chaining idea for learning Jazz improvisations, and ended with a complex model that expanded during the process. We found the significant final results and outcomes of the Turing test as demonstrations for the potential that Markov chains have in learning musical improvisations.

An important conclusion that took shape during the project is the high importance that our model constraints had on the results. Only throwing data on the primary algorithm didn't give us much, but as we adapted our model with heuristics, exploration, and stochasticity the results became more and more satisfying. The data and the markov chain itself weren't enough and the musical constraints themselves weren't enough, but the combination of the two together gave us optimal results. Our model assessments let us understand which of our model parameters had the highest significance. We found the Markov chain order to be a critical parameter on the final results.

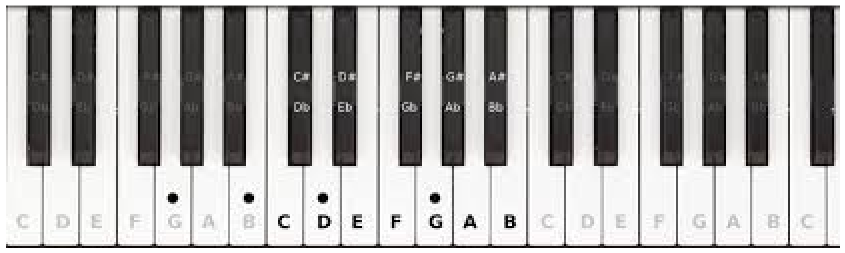
In the beginning of the paper we raised the question whether an algorithm can succeed in such an assumably emotional act as generating a musical improvisation? We found our results to be a proof of concept for the potential that an algorithm has in producing such melodies. However, music includes many more aspects besides the melody itself. That is why we wanted to suggest ideas for more work that can be done to improve our work in the future:

1. Adapting the model to learn chord progression, and improvising on it.
2. Considering the velocities of the notes being played
3. Considering the timbre of the notes being played and generating a melody on a different format than midi.
4. Adapting the model to learn harmony.
5. Combining use of different instruments when generating the piece.

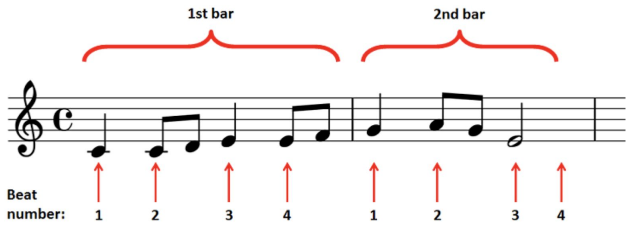
Appendix 1:

Basic musical conventions and concepts used in the paper:
A musical **melody** is a linear succession of musical notes. Notice the difference between a melody and **harmony**, which consists of the use of more than one note being played together. When we infer to a melody, we mean one line of **note** successions and their **rhythms**.

A Note, is each key you have, for instance, on a piano.



Rhythms, are the musical patterns in time. In music, we split the rhythms into musical **bars.** Each bar is basically a small time segment.



In the image above you can see two musical bars, containing different notes played on different lengths. The lengths of the notes are constantly changing, which gives us the rhythm of the melody.

Time segments in music are referred as **beats**, the pulse of the melody.

A slightly more complex concept of rhythms is the **metre** which implies the number of beats we have in each bar. For our project, the metre was eventually not relevant as we learned only the notes from the raw data (without their rhythms). The melody we output was played in one constant most common metre: 4/4 (meaning 4 beats in a musical bar)

The notes of the melody are often played in a certain **scale.** A scale is any set of musical **note ordering**. Different melodies are played on different scales, and some of them are played on the same scale. Two common types of westren scales are the Major scale and the Minor scale.

Each Major scale has a **parallel scale** which is a Minor scale, meaning two scales that use the same notes.

Another common type of scale is the **pentatonic scale**. The scale is specifically known for its complete **tonality**. Tonality generally infers to the relations between notes and chords on the scale. When we infer to a melody being **tonal** in the paper we mean that it uses stable relations between notes.

In harmony, **intervals** are the differences between two notes being played together. An **octave interval** in music notation is when we have the same letter for the note but on a different pitch. For example, lower C and higher C.

Another important element of harmony is chords. **Chords** are a set of notes playing at the same time. An **augmented seventh-chord**, common in jazz is a chord containing 4 notes played together, meaning a basic chord with an additional seventh interval.

<u>Scale transformation used in the project:</u>

In our project we learned from pieces played mainly on Major and Minor scales. When learning from different pieces we obviously need to shift the different melodies we learn from to be played on the same scale, so that the melodies will use the same set of note ordering. Doing so on the same type of scale is easy, for instance changing a melody from D major to C major requires shifting all notes from D major two notes down.

Synchronizing with Minor scales is also possible, because each Major scale has a parallel Minor scale, meaning two scales that use the same notes. For instance one can shift a melody played in B minor, two notes down to A minor, which is the parallel scale of C major. In this way you can have all your pieces synchronized, meaning that they use the same note ordering.

Appendix 2:

The full list of melodies we used during the project:

List of pieces in the Jazz group:

Lester young - dickies dream
Lester young - after theatre jump
Buck clayton - after theater jump
Bob berg - Second sight
Bob berg - Blues for bela
Bob berg - No moe
Charlie parker - embraceable you
Charlie parker - How deep is the ocean
Charlie parker - Yardbird Suite
Charlie parker - Koko
Louis armstrong - Basin street blues
Louis armstrong - Big butter and egg man
Louis armstrong - Cornet chop suey
Louis armstrong - Got no Blues
Louis armstrong - Gout bucky blues
Louis armstrong - Muskart Ramble
Louis armstrong - once in a while
Louis armstrong - Savoy blues
Johnny Dodds - Got no Blues
Johnny Dodds - Heebie Jeebies
Johnny Dodds - Hotter than that
Johnny Dodds - Muskrat Ramble
Johnny Dodds - Once in a while
Kid Ory - Got No blues
Kid Ory - Gut bucket Blues
Kid Ory - Muskrat Ramble
Kid Ory - Savoy Blues
Kid Ory - Who's it
Dickie Well - After theatre Jump
Dexter Gordon - Cheese cake
Dexter Gordon - Society Red
Dexter Gordon - Stanley the steamer
Buck Clayton - After theatre jump
Buck Clayton - Destination k
Buck Clayton - Dickies Dream
Art pepper - Desafinado
Art pepper - In a mellow tone
Art pepper - Stardust
Ben Webster - My ideal
Ben Webster - Bye bye blackbird
Benny Goldman - Tiger Rad
Benny Goldman - Whispering
Benny Goldman - Runnin Wild
Benny Goldman - Nobody's sweetheart
Benny `Goldman - Handful of keys
Benny Goldman - Avalon
Ray Charles - Georgia on my Mind
John Coltrane - So What
John Coltrane - Impressions
John Coltrane - Grand Central
John Coltrane - Blue Train
John Coltrane - Blues by Five
John Coltrane - Nutty
John Coltrane - Oleo
John Coltrane - Soultrane
John Coltrane - Countdown
John Coltrane - Trane's Blues
John Coltane - Mr. PC
Miles Davis - So What
Miles Davis - All Blues
Miles Davis - Blues by five
Miles Davis - Eighty One
Miles Davis - K.C Blues
Miles Davis - Oleo
Dave Brubeck - Take Five
Paul Desmond - Blue Rondo a la Turk
Chris Potter - Rumples
Chris Potter - Anthropology
Cannonball Addrley - So what
Wayne Shorter - Footprints

List of pieces in the funk group:

Herbie Hancock - Chameleon
Herbie Hancock - Dolores
Pepper adams - Early morning mood
Pepper adams - How high the moon
Joshua Rednam - I got you
Joshua Redman - Tears in heaven
Chris Potter - Togo
Kenny Garret - Brother Hubbard
Sonny rollins - Playin in the yard
Steve coleman - Cross fade
Woody shaw - Rosewood
Stevie Wonder - Higher Ground