A Novel Learning Automata-Based Strategy to Generate Melodies from Chordal Inputs

I. Helmy and B. John Oommen(✉)

School of Computer Science, Carleton University, Ottawa K1S 5B6, Canada
ibyhelmy@gmail.com, oommen@scs.carleton.ca

Abstract. This paper deals with the automated composition of music. Although music within the field of AI has been studied relatively extensively, the arena in which we operate is, to the best of our knowledge, unexplored. When it concerns computer composition, a noteworthy piece of research has involved the automated generation of the chordal notes when the underlying melody is specified. The chordal notes and beats have been generated based on models of well-known composers like Bach, Beethoven, and so on. The problem we study is the converse. We assume that the system is provided with the chords of some unknown melody. Our task is to generate a melody that flows with the given chords and which is also aesthetically and musically fitting. As far as we know, there is no research that has been reported to solve this problem and in that sense, although we have merely provided "baby steps", our work is pioneering.

Keywords: Learning Automata (LA) · Learning systems · Music composition · Markov chain model · Markov property

1 Introduction

From the time-immemorial theory of music, there has been a well-studied pattern for generating fundamental chords for a specific melody. This depends on the key in which the song is played, and the basic set of chords associated with this key are well documented. Thus, for a song in a particular key, for the most part, simplistic chordal arrangements can be generated, using a relatively small subset of chords associated with the key[1].

The problem that we consider is of a much higher dimension. The reader should observe that a sequence of chordal formations could, in turn, be suitable

[1] This is, of course, a very simplistic model. Indeed, the "majesty" of the composer is displayed by the wealth and spectrum of the chords that he uses.

B. John Oommen—Chancellor's Professor; Life Fellow: IEEE and Fellow: IAPR. This author is also an *Adjunct Professor* with the University of Agder in Grimstad, Norway.

Published by Springer Nature Switzerland AG 2020
I. Maglogiannis et al. (Eds.): AIAI 2020, IFIP AICT 583, pp. 203–215, 2020.
https://doi.org/10.1007/978-3-030-49161-1_18

for multiple melodies. Thus, even though the key of a song is known, and the chordal arrangements are specified, these chordal arrangements could fit to blend well for a myriad of underlying melodies. In other words, unlike the work that has been reported in the literature, our goal, which is the converse of the prior art, is to automatically generate melodies for chordal formations and sequences.

As human beings, we typically associate a musical piece with its primary melody. For example, while Mozart's clarinet concerto is accompanied by the orchestra, the melody is primarily played by the clarinet. Clearly, the problem that we study is fascinating, because we are attempting to generate a primary melodic tune that can be associated with the accompanying chordal music. This is by no means trivial, and even if we do accomplish such a goal, it will be the task of the musically-minded ear to decide whether the melody that has been generated is aesthetically beautiful. Although determining this is beyond the scope of our research, we have developed a AI-based system to achieve this.

AI has numerous tools such as problem solving, searching, Neural Networks (NNs), classification, and many more. The tool that we will use to achieve our goal forms the skeleton of reinforcement learning and is known as Learning Automata (LA). The beauty of using LA to solve this problem is that the action that the LA takes is chosen stochastically. The goal, of course, is to minimize a certain penalty function. If the actions of the LA are used to represent the musical notes, the output can be the melody sought for. Further, if the "Teacher" to the LA is, in some way, governed by the input chordal pattern, we can envision a scheme by which the output (the composed melody) fits in line with the provided chords. We believe that we have achieved this, our goal, to some extent[2].

1.1 Prior Art for Automated Music Composition

The two earliest notable works done on automated music composition appeared in 1988. The first, due to Lewis, [5], created an multi-layer Perceptron NN trained to form a certain desired mapping using a gradient-descent scheme. The second approach by Todd [4] used an auto-regressive NN to generate music sequentially. The computations of both these systems were limited by the hardware of that generation. Consequently, neither of these works could really succeed in taking their research any further. A common problem characterized by NN-composed music is that it often lack structure. The music generated in this way is limited in scope, and it thus does not possess a method by which it can represent the musical sequences created a few iterations before the current note.

The next notable work was done by Eck and Schmidhuber [6]. Their work aimed to address the lack of structure that the previous works suffered from. They proposed the Long Short-Term Memory (LSTM) algorithm, which was successful in the composition of well-structured and timed music that did not sway from its desired skeleton. Further, Eck who leads an AI team at Google under its *Google Brain*, also worked on the Magenta Project [7], which aims to

[2] A version of this system can be currently demonstrated and can be made available to the reader.

analyze the roles of machine learning in different aspects of the artistic world. Among the developments that emerged from the Magenta Project are Magenta Studio, which works with plugins and tools to be utilized with other music composition programs to generate and modify music. The Magenta project should also be credited for MusicVAE [8], which is a tool that takes two musical melodies and blends them together.

1.2 Learning Automata (LA)

We now briefly review[3] the field that we shall work in, namely, that of LA. The concept of LA was first introduced in the early 1960's in the pioneering by Tsetlin [22]. He proposed a computational learning scheme that can be used to learn from a random (or stochastic) *Environment* which offers a set of actions for the automaton to choose from. The automaton's goal is to pick the best action that maximizes the average *Reward* received from the *Environment* and minimizes the average *Penalty*. The evaluation is based on a function that permits the Environment to stochastically measure how good an action is, and to thereafter, send an appropriate feedback signal to the automaton.

After the introduction of LA, different structures of LA, such as the deterministic and the stochastic schemes, were introduced and studied by the famous researchers Tsetlin, Krinsky and Krylov in [22], and Varshavskii *et al.* in [23].

LA, like many of the Reinforcement Learning techniques, has been used in a variety of problems, mainly optimization problems, and in many fields of AI. Among other applications, it has been used in NN adaptation [11], solving communication and networking problems [12,14,16] and [17], in distributed scheduling problems [19], and in the training of Hidden Markov Models [10].

Figure 1 displays the general stochastic learning model associated with LA. The components of the model are the *Random Environment* (RE), the *Automaton*, the *Feedback* received from the RE and the *Actions* chosen by the LA.

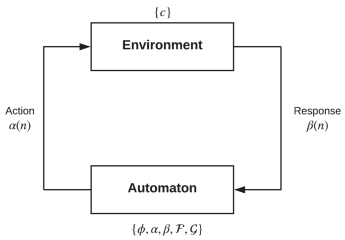

Fig. 1. The Automaton-Environment Feedback Loop.

The Environment: At every time instant, the LA picks an action from the set of actions available to it and communicates it to the RE. The RE evaluates that action as per a random function, and responds with a *Reward* or a *Penalty*.

[3] We will not go through irrelevant details and/or the proofs of the LA-related claims. They can be found in [24] and the reference book by the pioneers [13].

The Automaton: The LA achieves its learning process as an iterative operation that is based on the RE and their mutual interaction. The RE first *evaluates* the policy, which is how it appraises the selected action. It then undertakes *policy improvement*, by which it modifies the probability of selecting actions so as to maximize the *Reward* received from the RE.

Estimator and Pursuit Algorithms: The concept of Estimator Algorithms was introduced by Thathachar and Sastry in [20,21] when they realized that the family of Absolutely Expedient algorithms would be absorbing, and that they possessed a small probability of not converging to the best action. Estimator algorithms were initially based on Maximum Likelihood estimates (and later on Bayesian Estimates). By doing this, the LA converged faster to the actions that possessed the higher reward estimates. The original Pursuit Algorithms are the simplest versions of those using the Estimator paradigm introduced by Thathachar and Sastry in [20], where they *pursued* the current best-known action based on the corresponding reward estimates. The concept of designing ϵ-optimal discretized Pursuit LA was introduced by Oommen and Lanctot in [15].

2 Fundamental Concepts of Music Theory

Before delving into the details and structure of the design and creation of our AI module, in the interest of completeness, we briefly state some of the fundamental concepts[4] of the Theory of Music [1]. The most elemental term is that of a **note**, which is the most primitive unit that music is composed of. More specifically, a note is a symbol to represent sounds or pitches. Music is composed of notes, and in music, there are a collection of 12 notes in what is known as the **12-Tone Note Scale** depicted in Fig. 2 below.

C, C♯ | D♭, D, D♯ | E♭, E, F, F♯ | G♭, G, G♯ | A♭, A, A♯ | B♭, B

Fig. 2. The 12-Tone Note Scale

A **scale** is a subset of notes that correspond to a particular pitch. The two main scales used in this project are the Major and the Minor scales. A **chord** is a collection of two or more notes played together simultaneously. In the interest of simplicity and consistency, within the scope of this project, all referenced chords will be assumed to be triads, where a **triad** chord is one made up of exactly three notes. Thus, an "A Major" triad is composed of the notes: A, C Sharp, E.

The main difference between the two scales is the pitch, and this marginally affects the notes that constitute the chords being used. Thus, in the context of the above example for the "A Major chord", one observes that it is made up

[4] The terms and concepts which concern music are very fundamental, and are really included only in the interest of completeness.

of the notes: A, C Sharp and E. If one starts from the root note 'A', a Major chord moves 4 notes (3 to the right and then back again at the start to C) down the scale to the C Sharp note, and then 4 more notes to 'E' that completes the chord's formation. The "A Minor" chord, however, is composed of the notes: A, C and E. Here the middle note is increased by 3 notes rather than 4, which is essentially the difference between the Major and Minor chords. Finally, a **bar** defines a segment of music based on the tempo (number of beats per bar). Throughout the project, we assume a tempo of 120 bpm (beats per minute). Hence the bars will be 8 beats long, denoted as (4, 4) musically.

2.1 The Circle of Fifths

Our goal is to have an AI program learn to create melodies for chord progressions that are musically coherent. This means that through studying the input music, it will be able to determine which chords typically come after others, and hence form satisfying sequences. The tool that aids musicians in this task is known as the "Circle of Fifths", where as displayed in the figure below, the chords are all placed side-by-side in a circular fashion [18] (Fig. 3).

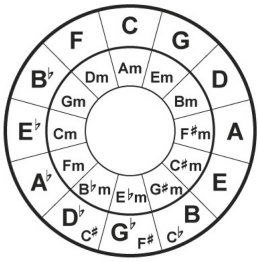

Fig. 3. A diagram displaying the Circle of Fifths

The circle creates a mapping for 12 chords ordered in such a way that when deciding what chord should follow the current one, one can moving down the circle in either direction to infer which chord will provide the best sounding match. The further away that a chord is in the Circle, the less likely the chord switch will satisfy the melody, which is central to creating cohesion in musical pieces. However, it is sometimes used as an artistic tool to generate a sense of "discomfort", and the use of such a device is left to the discretion of the artist.

2.2 Keys

Keys are a collection of chords that all fit within the same tonal basis as each other. Throughout this project, we will only consider melodies and chords within the Key of C Major. The triad chords in the key are as follows [2]:

- i – C major, (C)
- ii – D minor, (Dm)
- iii – E minor, (Em)
- iv – F major, (F)
- v – G major, (G)
- vi – A minor, (Am)
- vii – B diminished, (B)

2.3 Assumptions

Since we are attempting a *prima facie* solution, we make a few simplifying assumptions and restrictions namely, that the key in which the music will be handled and created is the Key of C Major, and that all the chords in the progressions are exactly *one* bar long. We also assume that each bar will be limited to exactly *four* notes, and that are all of even duration as well.

3 The Markov Chain

Fundamentally, music is based on patterns, and more specifically, patterns of sounds that mimic emotions and particular feelings. This is what renders music to seem to be uniquely human. To formalize how we could teach this phenomenon to an AI program, our hypothesis is that we have to relay the non-randomness aspects of "music" as a concept. However, the art of *writing* music is, in itself, a connected and partly dependant journey, implying that any single part of a song could be heavily reliant on what was composed earlier. However, one must still retain an important sense of independence between the notes and sections. Our model is that structure can be enforced or achieved in a formulaic manner, but the flow of music has to be taught with a certain sense of spontaneity. Thus, we consider stepping away from traditional progressions every now and then, thus adding to the "spice" that music contains. We achieve this by modeling the process using a Markovian Model so as to provide a mix of randomness which, in turn, is based on the probability aspects of the music that has been established.

A Markov chain is a mathematical system built to model statistical probabilities of random processes. These processes move from one state to another while satisfying a particular property, which states that the probability of a particular transition from any state to another is completely independent of any past sequence of states that preceded it. This is particularly useful in Computer Science, since there would then be no need to store large amount of information.

A Markov model follows the following formula:

$$Pr(X_{n+1} = a | X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) = Pr(X_{n+1} = a | X_n = x_n).$$

This specifically deals with a sequence of previous states $X_1, X_2, \ldots, X_n$, where the current state is X_{n+1}. The formula demonstrates that moving from one of these states to another state is solely dependent on the most recent state, since $Pr(X_{n+1})$ is only based on of the probability of moving from the state X_n.

Figure 4 depicts a state diagram for three chords; C major, D minor, E minor.

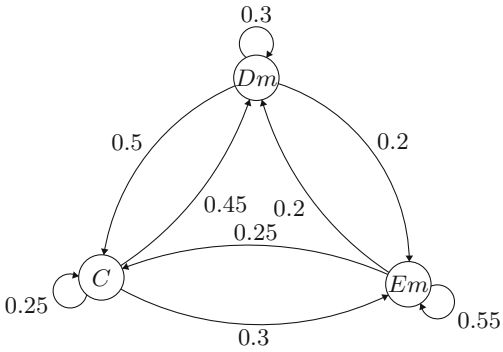

Fig. 4. An example of a Markov Chain for a set of three chords

Here, the transition matrix is a 3×3 matrix, where the chords on the left are the current chords, and the chords on the top are those to be transitioned to:

	C	Dm	Em
C	0.25	0.45	0.3
Dm	0.5	0.3	0.2
Em	0.25	0.2	0.55

3.1 The Model for Chords and Melodies

The probabilities of the state transitions are maintained in a stochastic matrix, the *transition matrix*, whose row sums are unity. Indeed, one can observe that from any given chord the possible state transitions sum up to unity.

Because there are 7 chords within the key of C, the matrix is a 7×7 matrix, and the $[i, j]$ entry is the probability of switching from chord 'i' to 'j'.

To help with the book-keeping, we keep track of which changes are mapped, and to the corresponding chords. To do this, we keep an identical matrix, the "Name" matrix, which contains the two chord symbols rather than the probabilities at the same entry. For example, in the transition matrix, the entry [0,1] has probability 0.4. In the "Name" matrix, entry [0, 1] contains the string "CDm" implying that chord changes C to Dm occurred with probability 0.4.

Analogously, we maintain a similar matrix for the melodies. However, the novelty of our strategy is that instead of mapping the notes to each other, like the chords, the notes are mapped to the *chords* being played. Modeling the matrix in this way allows the AI module to possess a greater control over the notes being played. This matrix is, therefore, also an 7×7 matrix where each chord contains the probabilities that a note is played alongside it in a song. Observe that as per the model, a random *note* is chosen based on that particular probability distribution depending on the *chord* selected.

Code Implementation Details: We now specify the details of the Markov Model's code implementation. This entails declaring the model's entire state-space, which are the 7 triad chords of the key of 'C'. We thus define the transition matrices and the "Name" matrices, where the probabilities of the former are initialized uniformly with the exception of the last transition which is slightly higher than the rest. There is no particular reason for this except for the fact that one cannot achieve a even distribution of probabilities among 7 numbers. Besides, we know that the chord of 'B' is used minimally in the key of 'C', especially in pop songs. The matrices for the chords and melodies are similar, and a typical example is given below for the chords.

```
transitionMatrix =
[[0.14, 0.14, 0.14, 0.14, 0.14, 0.14, 0.16],
[0.14, 0.14, 0.14, 0.14, 0.14, 0.14, 0.16],
...
[0.14, 0.14, 0.14, 0.14, 0.14, 0.14, 0.16],
[0.14, 0.14, 0.14, 0.14, 0.14, 0.14, 0.16]].
transitionName =
[['CC', 'CDm', 'CEm', 'CF', 'CG', 'CAm', 'CB'],
['DmC', 'DmDm', 'DmEm', 'DmF', 'DmG', 'DmAm', 'DmB'],
...
['AmC', 'AmDm', 'AmEm', 'AmF', 'AmG', 'AmAm', 'AmB'],
['BC', 'BDm', 'BEm', 'BF', 'BG', 'BAm', 'BB']].
```

We also need the imports relevant to the Markov Model, namely, `numpy`, the package for scientific computing in Python, as "np", and random as "rm".

The helper functions (and their tasks) that we define are summarized below:

- `load`(*str*): This function takes a song file name and loads it into the program;
- `getTransIndex`(*str, matrix*): Gets the index of a certain transition from the transition matrix passed in;
- `getShorthand`(*chord*): Returns a list of the notes contained within a *chord*;
- `adjustMatrix`(*change*, *degree*): Adjusts the transition matrix probability for a certain chord *change* by a certain *degree* (percentage);
- `adjustMelodyMatrix`(*melodyDict*): For every chord in the dictionary, this function adjusts the probabilities of the notes played over it;
- `convertNotes`(*notes*): Takes the data from the MIDI files and converts them into notes which are to be used in the learning process;
- `convertChords`(*chordz*): Takes the data from the MIDI files and converts them into chords to be used in the learning process;
- `normalize`(): Normalizes the values in the transition matrix to ensure that it retains its stochastic property after any adjustments;
- `formVerse`(*bars*): This function yields a verse structure to be filled with chords, where a sequence of *bars* is passed in;
- `formChorus`(*bars*): The function yields a chorus structure to be filled with chords with a sequence of *bars* passed in.

The Chord Creator: The final part of the code to be discussed is the function `chordCreator`(*bars, sChord*). This is the module where the Markov Model itself is implemented for the chord progressions. To initialize the module, we specify the starting chord selected based on the input passed into the function. This, is, in turn, displayed to the user. This initializes an empty list containing only the starting chord. Thereafter, we create a counter used to track how many bars have been created so far. The largest chunk of the function deals with the selection of the next chord. Using the `numpy` and `random` libraries, the next chord is selected based on the Markov Chain, and added to the list of chords. The function used to construct the chords is:

`np.random.choice`(a, size=None, replace=True, p=None)

The above function takes 4 parameters. The first, 'a', is a list of possible elements from which we can generate a random sample. The second parameter defines the shape of the output, if needed. The next is a boolean variable, labeled 'replace', which accounts for the fact whether the sample is generated "with" or "without" replacement. The final parameter is the quantity 'p', which specifies the probabilities associated with each entry in the chain. In our case these parameters are as follows: We pass the list of chord transitions for that particular starting chord as the parameter 'a', we place no size restriction, we set the variable 'replace' to be *True*, and we finally set 'p' to be the chain's transition matrix.

The code is designed to handle 7 individual cases, one for each possible current starting chord. For each of the possibilities, the random choice function (discussed above) decides on what chord the system transitions into. Once this is done, the new chord is set as the starting chord and appended to the list of chords contained within the chord progression so far (i.e., chordList).

After the next chord in the progression is decided on, it is set as the new starting chord, and the counter is incremented. Once the number of bars is completed and the while loop terminates, the program yields as its output a chord progression along with a probability value of that specific chord sequence.

The code for the "Melody Creator", `melodyCreator`(), is quite similar. However, for each chord, the program selects four notes to be played. This is due to the previously-stated assumption that each note lasts for a quarter of a bar, and that a chord is played for a whole bar. Again, the `np.random.choice` function is used to determine the note that is played. Once verified, the note is added to the list of notes, and once the code terminates, the function returns the list of notes played, and the associated probability of the created melody.

4 The Learning Process

We execute reinforcement learning by means of a LA specifically constructed for this purpose. The actions that are chosen as based as per a specific probability distribution, which in our case is the Markov Model previously described. In our case, however, we deal with two parallel LA, the first of which deals with the learning involved with the chords, the second with the melodies. The interactions between the LA and the RE (the music-related data) is explained below.

4.1 Learning the Chords

To learn to generate chord progressions, the AI program is fed with the appropriate form of data. To achieve this, we went through a list of songs in the key of C major. After selecting the songs, we invoke the website `ChordsWorld.com`, to retrieve the chords used for each song. Once the chords for each song are collected, they are placed into .txt files, where the chords are separated by tabs. For the parsing of the chords data, we use a "load" function which opens the file and parses it based on the tab spacing. Once the file is parsed, the list of chords is returned by the function to be used in the learning process.

Before proceeding, the code examines the two cases where the current chord or the next chord are both not those commonly used for the key of C major. If this occurs, the code skips these chords and resumes the learning. With every chord change made, we resort to a call to the `adjustMatrix` function, after which the chord is added to the list of chords. After all the adjustments are made, we normalize the rows so as to maintain the stochastic property of the matrix.

4.2 Learning the Melodies

It is far more difficult to find a large public source of sheet music for the melodies of the songs presented and "taught" to the AI program. Consequently, we created short song clips written by the First Author using a Digital Audio Workstation (DAW) known as `FL Studio`. Each mini-song utilizes the chords from the key of C and has a very different set of melodies. By connecting it to a MIDI keyboard, one could play the music directly.

As is well known, a .mid file (MIDI file) is a form of an audio file that contains many channels. When using `FL Studio`, we write the chords onto one channel and the melodies onto another. We then combine the two into a single .mid file for learning. The .mid files were read using the Python MIDI library, readily available on Github [3], using which we are able to open and read the different channels and to extract the pitch data. Thus, we are able to analyze the pitches to determine the chords/notes being played using the `convertNotes` and `convertChords` functions (Fig. 5).

The above snippet of code shows how the pitch data from the chords passed in, is translated into the actual chord names. The list is traversed in increments of three, since the chords are all triads, and only the first note is needed to be able to identify the chord. This process is again repeated for the `convertNotes` function, but this time every note is converted and stored.

In order to adequately judge the interaction with the Environment when it comes to the melodies, we use a different approach than what we is done with the chords. For each chord played, we set a key in a python dictionary. From there, we add a value under that key for every note played alongside that chord. This is always the next four notes, as per the assumption that for every bar of music there is a single chord and four notes. Once this is accomplished, we iterate through the dictionary for the matrix of note probabilities and adjust the matrix entries as needed using the `adjustMelodyMatrix` function. Again,

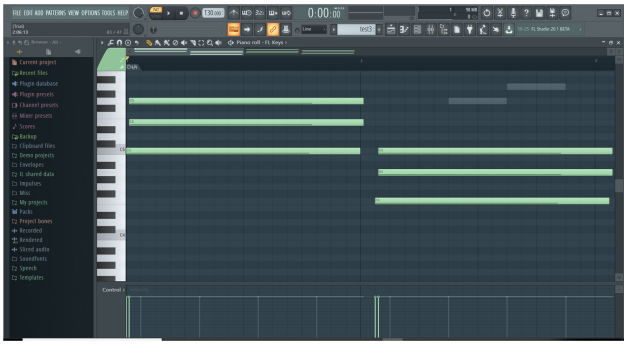

Fig. 5. Example of an FL Studio Session

after all the adjustments are made, we invoke a call to the `normalize` function to ensure that stochastic property of the matrix is maintained.

5 Results and Discussions

Understandably, it is not possible to adequately discuss the audio results that we have obtained in a paper document. Indeed, a prototype solution for the system that we have proposed is available, and a version of this system can be currently demonstrated. This prototype can be made available for interested listeners.

To produce the final output we used another library called `midiutil`. Using this, we were able to write the music that the AI system created into MIDI files. The chords and notes generated were passed into the `writeMusicOutput` function where the chords were first written on a channel and the corresponding notes on another, after which both were written to the same file and saved so as to be played by any basic media player that could play .mid files. The entire project was written and executed on Windows 10, and in it we used the native Windows Media Player application to listen to all the music that was produced. As mentioned earlier, while most of the other works in the field of music composition using AI were focused strictly on making music, our approach focused more on the generation of melodies based on the chords that were also generated. Thus, given a sequence of chords, the AI was capable of improvising a coherent melodic structure as an accompaniment. Although our solution is a *prima facie* prototype, we have succeeded in the task that we undertook.

6 Conclusions

In this paper, we have dealt with the automated composition of melodic (as opposed to chordal) music. Although music within the field of AI has been studied relatively extensively, the problem that we have studied is, to the best of our knowledge, unexplored. The existing research when it concerns computer

composition, has involved the composition of the chordal notes when the underlying melody is specified, where these chords and beats have been generated based on models of well-known composers. The problem studied here is the converse, where we assume that the system is provided with the chords of some unknown melody. Using the foundations of Markov Chains and the tools of Learning Automata, we have been able to generate a melody that flows with the given chords. The melody generated is aesthetically and musically fitting. A prototype system that achieves this is currently available, and as far as we know, there is no research that has been reported to solve this problem. Although we have merely provided a primary solution for this rather complicated endeavor, we believe that our work is pioneering.

References

1. Khan Academy: Glossary of Musical Terms (2018). https://www.khanacademy.org/humanities/music/music-basics2/notes-rhythm/a/glossary-of-musical-terms
2. Cazaubon, M.: Chords in the Key of C Major (2018). http://www.piano-keyboard-guide.com/key-of-c.html
3. vishnubob: pyhton-midi (2013). https://github.com/vishnubob/python-midi
4. Todd, P.: A sequential network design for musical applications. In: Proceedings of the Connectionist Models Summer Schooli (1988)
5. Lewis, J.: Creation by refinement: a creativity paradigm for gradient descent learning networks. In: International Conference on Neural Networks (1988)
6. Eck, D., Schmidhuber, J.: Finding temporal structure in music: blues improvisation with LSTM recurrent networks. In: IEEE Workshop on Neural Networks for Signal Processing (2002)
7. Google Brain: Magenta (2018). https://ai.google/research/teams/brain/magenta/
8. Google Brain: MusicVAE: creating a palette for musical scores with machine learning (2018). https://magenta.tensorflow.org/music-vae
9. Ghaleb, O., Oommen, B.J.: Learning automata-based solutions to the multi-elevator problem (2019, to be Submitted)
10. Kabudian, J., Meybodi, M.R., Homayounpour, M.M.: Applying continuous action reinforcement learning automata (CARLA) to global training of hidden Markov models. In: 2004 International Conference on Information Technology: Coding and Computing. Proceedings, ITCC 2004, vol. 2, pp. 638–642. IEEE (2004)
11. Meybodi, M.R., Beigy, H.: New learning automata based algorithms for adaptation of backpropagation algorithm parameters. Int. J. Neural Syst. **12**(01), 45–67 (2002)
12. Misra, S., Oommen, B.J.: GPSPA: a new adaptive algorithm for maintaining shortest path routing trees in stochastic networks. Int. J. Commun. Syst. **17**(10), 963–984 (2004)
13. Narendra, K.S., Thathachar, M.A.L.T.: Learning Automata: An Introduction Prentice-Hall, New Jersey (1989)
14. Obaidat, M.S., Papadimitriou, G.I., Pomportsis, A.S., Laskaridis, H.S.: Learning automata-based bus arbitration for shared-medium ATM switches. IEEE Trans. Syst. Man Cybern. Part B (Cybern.) **32**(6), 815–820 (2002)
15. Oommen, B.J., Lanctot, J.K.: Discretized pursuit learning automata. IEEE Trans. Syst. Man Cybern. Part B (Cybern.) **20**(4), 931–938 (1990)
16. Oommen, B.J., Roberts, T.D.: Continuous learning automata solutions to the capacity assignment problem. IEEE Trans. Comput. **49**(6), 608–620 (2000)

17. Papadimitriou, G.I., Pomportsis, P.A.S.: Learning-automata-based TDMA protocols for broadcast communication systems with bursty traffic. IEEE Commun. Lett. **4**(3), 107–109 (2000)
18. Paszakowski, S.: The Circle of Fifths: The Clock of Key Signatures (2019). https://www.libertyparkmusic.com/the-circle-of-fifths/
19. Seredyński, F.: Distributed scheduling using simple learning machines. Eur. J. Oper. Res. **107**(2), 401–413 (1998)
20. Thathachar, M.A.L., Sastry, P.S.: A new approach to the design of reinforcement schemes for learning automata. IEEE Trans. Syst. Man Cybern. **SCM-15**(1), pp. 168–175 (1985)
21. Thathachar, M.A.L., Sastry, P.S.: A class of rapidly converging algorithms for learning automata. In: IEEE International Conference on Systems, Man and Cybernatics. IEEE (1984)
22. Tsetlin, M.: On behaviour of finite automata in random medium. Avtomatika i Telemekh **22**(10), 1345–1354 (1961)
23. Varshavskii, V., Vorontsova, I.P.: On the behavior of stochastic automata with a variable structure. Avtomatika i Telemekhanika **24**(3), 353–360 (1963)
24. Zhang, X.: Advances in the theory and applications of estimator-based learning automata. Ph.D. thesis, University of Agder, Norway (2015)