AI Encountering Interactive Systems: A Deep Learning Reinforced Musical Composition Model

Ke Ma  
College of Design and Innovation, Tongji University  
Suri and Aurora Inc.  
suta@tongji.edu.cn

Tian Xia  
School of Electronic Information and Communications,  
Huazhong University of Science and Technology  
tianxia@hust.edu.cn

ABSTRACT

In this paper, we present an artificial intelligence-based musical composition algorithm for generating harmonic and various arpeggios based on given chords in real-time, which incorporates a recurrent neural network (RNN) with the gated recurrent unit (GRU) and a tetrahedral context-sensitive L-system (TCSL). RNN can play a significant role of learning inherent harmony from arpeggio datasets and providing probabilistic prediction which suggests the selection of the next note and the lengths of the selected notes. We also establish TCSL model based on a tetrahedron model utilizing seven interval operators and production rules to increase the variety of generation. TCSL is responsible for generating one note at each iteration by respectively requesting probabilistic predication from RNN, calculating optional notes and determining target note. Our experiments where we trained two RNNs for TCSL generation model indicated that the proposed algorithm has advantages in overcoming the obstacles of achieving inherent global harmony as well as the variety of generation in current computer-aided arpeggio composition algorithms. Our research attempts to extend deep learning model (DLM) towards the design space for interactive composition systems.

1. INTRODUCTION

The advent of computer-aided algorithmic composition has enabled a novel approach for generating music without using real instruments. For instance, Repons [33] applied this method to implement a computer-assisted arpeggiator in improvisation. With this concept, many studies developed arpeggio generation systems performed in the real-time scene [26].

The development of arpeggio generation systems is an ongoing work with an increasing demand for musical harmony and variety. However, two main obstacles are lying in such systems: (1) the note generation of current algorithms are mostly conducted with a stochastic and intuitive way, which lacks inherent global harmony. However, the connections between the previous notes and the next note are supposed to be relevant in global rhythmic structure. To achieve the harmony, the computer-aided algorithms need to learn global patterns from the existing arpeggio datasets using a machine learning model. New articles are composed according to these learned harmonic and rhythmic structures; (2) the output generation of current machine learning-based algorithms, nevertheless, have limited variety because they begin typically with an initial seed but the number of initial seeds is insufficient. For example, in arpeggio generation, only tonal notes of each chord spanned in multi-octaves can serve as the starting note, which means the output is confined, lacking variety.

To tackle the obstacles, we developed an artificial intelligence-based musical composition algorithm incorporating a recurrent neural network (RNN) and a context-sensitive L-system [35]. We introduced gated recurrent unit (GRU), a type of recurrent unit in recurrent neural networks (RNNs) [9], which was recently proposed to capture long-term dependencies in a sequence by Cho et al. [1]. It manifests an advantage of adaptively remembering and forgetting its memory contents based on the input data to the unit [2]. Therefore, RNN is responsible for learning patterns from existing musical datasets and providing the probabilistic prediction of the next output and the lengths of the notes.

In addition to RNN, we established a tetrahedral context-sensitive L-system (TCSL), serving as the generation model, capable of reinforcing the variety of generated composition. The proposed TCSL incorporates various tetrahedral generation variants constructed by notes and interval operators [19]. TCSL model generates one note at each iteration by respectively requesting probabilistic predication from RNN, calculating optional notes and target note.

With the two core algorithmic components, our algorithm can generate global harmonic and various arpeggios along with real-time chord progressions. A collection of generated arpeggio pieces by the system is assembled on this paper’s accompanying website 1 for audiences for better understanding our algorithm. Further, the method in this paper can be extended towards a design space for interactive composition systems where users can

1 http://www.signichat.com/suta/arp/

Copyright: © 2018 Ke Ma, Tian Xia. This is an open-access article distributed under the terms of the Creative Commons Attribution License 3.0 Unported, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.
manipulate music and interact with deep learning model (DLM) through tangible user interface (TUI).

The remainder of the paper is organized as follows: Section 2 introduces the algorithm-based arpeggiators and music generation networks. In Section 3 we evaluate the RNNs and the output of the algorithm. Finally, Section 4 presents conclusions and the future work.

2. BACKGROUND

2.1 Algorithmic Arpeggiator

As a harmonic and rhythmic melody, the arpeggio is widely used in improvisation and real-time interactive composition systems. With algorithmic composition methods, arpeggios can be composed utilizing computer-aided software "arpeggiator" along with the chord inputs. Here we briefly describe an evolution of algorithm-based arpeggiators applied in improvisation and real-time interactive composition systems.

The development of arpeggiators can be categorized into two phases. In the first period, the arpeggiator can only support linear/one-dimensional inputs. Risset et al. [26] devised an acoustic piano accompanied by a computer in 1996. A pianist played the same arpeggio three times, while the system played three different mirrored accompaniments. The pianist’s performance determined the tempo and loudness of the arpeggio. This has established the generation pattern of the algorithmic arpeggiators, mapping from input to output.

The following aquarium project [14] allowed users to generate major chord arpeggios by mimicking the movement of animals through a tangible object. In Princeton Laptop Orchestra [28], each player was provided with a controller to modulate the overtones and the fundamental frequency of Risset Arpeggio [27], improvising the beating patterns. At this point, the ascent and descent of arpeggio were coordinated by direct inputs.

In the second phase, multi-functional sensors and actuators were integrated into algorithmic composition systems and therefore arpeggiator becomes capable of accommodating complex nonlinear/multi-dimensional inputs. For example, these devices can capture users’ physical behaviors and convert them as system inputs so that information sent to arpeggiator become much more synthetic than previous systems. Sonic Banana [29] mapped multi-channel data from four sensors to various parameters of algorithmic arpeggiator such as chord type, note duration, velocity, root pitch, and so on.

Applications of human motion capture (HMC) [6] technology [20] shows another way for real-time composition. It utilizes physical movement to interact with the composition system. Therefore, the parameters of HMC-based arpeggios were inextricably tied to the physical gestures/postures or eye gazes of the performer. Gestate [21] allowed performers to trigger chords by rotating an arpeggiated 3D cube through right hand. The position of left hand otherwise reflected the glissando arpeggio. EyeHarp [34] provided a gaze-controlled musical interface for performers to customize parameters of four mixing arpeggios. Gazes were allocated to control the starting note, meter, notes included, and so on.

With the integrations of multi-dimensional inputs to computer-aided composition system, interactions with the arpeggiator grow more natural than as previous. The outcome of the algorithm is intimately associated with performer's physical movement and mental model. In this way, arpeggios are mostly conducted with a stochastic way.

2.2 RNN Music Models

RNNs are connectionist models that memorize dynamics of arbitrary sequences. RNNs can provide probabilistic prediction leveraging the trained models. Todd [31] proposed a pioneer work by utilizing an RNN model to generate music compositions on a note-by-note basis. The generated compositions are supposed to be musically coherent given that RNNs can capture temporal dynamics.

However, in practice, the lack of long-term memory of note sequences results in lacking global coherence which was discovered by Mozer [22]. The capability of storing long-term dependencies has diminished because of the vanishing gradients. To solve this problem, Mozer claimed to apply sophisticated RNN techniques and psychologically-realistic encoding methods [16, 30] to generate distributed representation of musical features. Nevertheless, the difficulties remain in extracting global thematic structure and rhythmic organization by means of RNN.

Hochreiter and Schmidhuber devised a Long Short-Term Memory Neural Networks (LSTMs) to overcome the vanishing gradients phenomenon [11]. The LSTM model raises a memory cell to substitute the ordinary nodes in standard RNNs. Constant error carousel, a self-connected recurrent edge, is involved in the memory cell, ensuring that gradients can pass across beyond 1000 time steps without vanishing or exploding [11, 17].

LSTMs operate by three functional gates to forget, memorize and expose the memory contents [2]. The input gate modulates how much relevant input can flow into the memory cell, while the forget gate introduced in [10] permits an LSTM to reset the obsolete memory [7] adaptively. The output gate determines how much relevant input can flow into the memory cell. However, the lack of long-term memory of note sequences results in losing global coherence which was discovered by Mozer [22]. The capability of storing long-term dependencies has diminished because of the vanishing gradients. To solve this problem, Mozer claimed to apply sophisticated RNN techniques and psychologically-realistic encoding methods [16, 30] to generate distributed representation of musical features. Nevertheless, the difficulties remain in extracting global thematic structure and rhythmic organization by means of RNN.

In practice, the novel model has presented an ability to learn long-term dependencies using the internal memory cell and functional gates. Further, Douglas and Lapalme introduced a music-specific sequence learner incorporating an LSTM model to learn global musical structure [8]. Among the most recent LSTMs-based musical systems,

2.3 Gated Recurrent Unit

GRU is a novel recurrent unit proposed by Cho et al. [1]. Similar to LSTM unit, GRU is designed to capture long-term dependencies using a gated activation function. Update gate and reset gate enable a memory cell to adaptively update and refresh memory contents [4]. Unlike LSTM, GRU exports memories to the next node without the supervision of an output gate.

By comparing new candidate input with the stored memories, update gate in GRU determines the acceptance of new candidate memories in memory cells. Update gate would not accept input features when captured features remain valuable to the cells in the next time steps. Otherwise, input features would be memorized and carried across multiple time steps. A reset mechanism is designed to improve its efficiency using the reset gate. It allows GRU to refresh the memory cells whenever the captured features are useless [2] (See Figure-1B for a graphical illustration of GRU memory block).

![Figure 1. Memory blocks of (A) LSTM and (B) GRU. (A) i, f and o are the input, forget and output gates, respectively. (B) r and z are the reset and update gates.](image)

Since GRU is recently introduced in music generation, some studies focused on the evaluation of GRU on the task of music modeling. Chung et al. empirically evaluated GRU and LSTM on a number of polyphonic music datasets [4]. Jozeefovicz et al. conducted a thorough exploration of over ten thousand RNN variants to identify a superior architecture [15]. LSTM and GRU respectively demonstrated superiority on different music datasets.

2.4 L-systems

An L-system is a robust approach for generating music compositions based on rewriting rules [24]. Context-sensitive L-systems were proposed to introduce variety [35] in music generation.

L-systems were firstly proposed for modeling plant development [25] and other iterative structures. Symbols grammatically constrain each iteration and rewriting rules. The rendering process can be graphically interpreted when distributing each symbol with a graphic command. After a few iterations, a graphical structure can be rendered.

However, not all L-systems are considering graphical interpretation. As music inherently follows grammars and the notion of iteration, it fits nicely in the context of rewriting that L-systems provide [18]. Therefore, some authors used L-systems to generate music [19, 23, 24].

In early stages, music scores are generated through graphical interpretation. Prusinkiewicz [24] mapped turtle drawing into musical scores, by mapping y-coordinates to pitch and line length to note durations. To the next stage, McCormack [23] used symbols to directly represent pitch, duration, and timbre without going via a graphical rendering, which allows researchers to concentrate on mapping algorithm between music parameters and L-systems grammars.

Although musical renderings are no longer constrained with graphical interpretation, the music notes generated with fixed production rules appear monotonous, which lacks variability. Regarding this drawback, Peter and Susan proposed two novel L-systems: stochastic and context-sensitive, respectively. Stochastic L-systems are used to generate music which has the similar motif, but different details [35], while L-systems with context sensitivity provided an approach to introduce variety [18]. In context sensitive L-systems, multiple production rules are applied to one symbol in a specific context [35]. It allows L-system to compare and select production rules depending on the surrounding symbols. In this way, the various melody is generated by selecting appropriate production rule along with iterations.

3. EXPERIMENT

To implement a musical arpeggio generation algorithm, we integrate RNN with TCSL for generating harmonic and various arpeggios. By taking advantage of RNN, internal harmony between chord notes are learned from existing pieces of the arpeggio. While TCSL is responsible for generating variety. Therefore, our system aims at overcoming the obstacles of lacking inherent harmony and variety in current arpeggiators.

The system takes arpeggio MIDI datasets as input and generates arpeggio sequences on different underlying chords. We conducted an experiment where we have trained an RNN model to provide probabilistic prediction which suggests the selection of next note along with the lengths of the notes for TCSL generation model. Our
system consists of two modules (1) RNN modeling (2) TCSL generation.

3.1 RNN Modeling

In this module, we used MIDI Arpeggio dataset\(^2\) purchased from Prosonic Studio\(^3\) as model inputs, which contains 2715 major and 2178 minor triad arpeggios. These MIDI arpeggios cover various phrase types and chord progressions. We randomly picked up the training dataset and test dataset by a ratio of 4:1 for each tonality. When extracting note sequences from MIDI data, we used a MIDI Python library \([12]\) to transform MIDI files into a two-dimensional matrix. The sampling rate fixed at 0.25 Hz. X-axis stored the length of a sequence, while y-axis stored an one hot vector with 88 notes. An initial element was appended to the end of the vector to predict the first note. The RNN model was implemented in Python using Keras library \([5]\). For major and minor keys modeling tasks, we respectively trained one RNN. RNN modeling aims at providing a probability distribution by maximizing the prediction accuracy.

The RNN took an one hot vector as input and predict which note will come up next after the previous notes. In each step of the note prediction, the network calculated a probability table, recorded probability of next appearance of each note, and selected the maximum probability as current output. The probability table served as the basis for choosing the following note. If the note chosen is equal to the previous one, the length of this note will be increased and therefore modeled by the network. By comparing the predicted result with the target data in the input sequence, RNN gradually improved the accuracy of prediction.

Our network consists of two hidden layers with GRU units and one full-connected output layer with tanh activation function. The number of hidden GRU blocks in the hidden layer was fixed at 250. We used RMSProp \([13]\) to optimize the model parameters. According to many experiments and results on the validation set, we selected a learning rate of 0.0005 to minimize the log-cost. We trained each model for 85 epochs because the total error had not improved as we increased the number of training epochs. The validation made use of early-stop training to prevent over-fitting. We also provide a reference implementation of the system as open source software on GitHub\(^4\).

3.2 TCSL Generation

With trained RNN model, we next proposed a novel tetrahedral context-sensitive L-system for generating a variety of arpeggios. TCSL incorporates a tetrahedron model. Each vertex of the tetrahedron stores a note and each side represents a type of interval operator. Because the tetrahedron inherently has three sides from an arbitrary vertex to the others, it constitutes a selective context for iteration. This model nicely fits for the context-sensitive L-systems where multi production rules are applied to one symbol \([35]\). The advantage of TCSL is that the tetrahedral model can evolve into a large number of variants. For example, TCSL-A and TCSL-B shown in Figure 2 are two variants established by different sets of operators. Each variant can generate a distinct output. Therefore, we created our TCSL based on this tetrahedron model to overcome the lack of variety. See Figure 2 for the illustration.

In this model, note stored in a vertex can be rewritten by a new note at each iteration. The operator on each side calculates the inherent intervals between current note and target note. One current note and three target notes constitute a specific context.

Firstly, we dealt with the notes and operators used for generation. Considering an arpeggio, 15 chord notes (I, III, V) spanned in 5 octaves were included in a sequence according to the tonality and chord. For instance, if given the Dmin chord, the arpeggio sequence would be \{D0, F0, A0, D1, A1, F1, D2, F2, A2, D3, F3, A3, D4, F4, A4\}. Two adjacent notes executed one interval. Encoded operators are:

+1: up one interval
+2: up two intervals
+3: up three intervals
-1: down one interval
-2: down two intervals
-3: down three intervals
^: flip horizontal

![Figure 2. TCSL-A and TCSL-B with different operators and initializations. C means current and t means target.](https://github.com/SutaBeacon/music_rnn)

Secondly, we explained the variety of generation in terms of the selection of operators and initialization. First, Operators are selected from seven defined ones and then distributed onto six sides, which constitutes 2520 \((7!/2)\) possible sets of operators. Besides, initialization by randomly selecting a starting vertex and a starting tonal note also increase the variety. Each set of operators and initial conditions can generate a unique output. Two representative samples are shown in Figure 2.
Finally, we introduced the workflow of TCSL generation model in this system. See Figure 3 for the schematic.

Figure 3. A schematic of TCSL generation showing (A) Input chord, (B) Start new iteration, (C) Request probability table from RNN, (D) Calculate optional notes, (E) Determine target note, (F) Generate arpeggio note. Red color denotes current note and blue indicates target note.

(A) Input chord. Whenever a chord in one key is imported into TCSL, a new round of generation starts off. Meanwhile, TCSL is initialized.

(B) Start new iteration. The new iteration starts from the vertex storing the target note in the last iteration. Target note is rewritten to current note.

(C) Request probability table from RNN. TCSL delivers the present note to RNN, requesting probability table for the next iteration. RNN calculates the table based on the complete sequence fed to the network including previously generated notes and currently received one. Then RNN sends the table to TCSL.

(D) Calculate optional notes. Optional notes consist of current note and three on the target vertexes. Given that RNN is capable of modeling the length of the note, current note here is used for determining if it has the chance to be continued in the next iteration. TCSL figures out the three notes according to current note and individual operator.

(E) Determine target note. Once receiving the probability table, TCSL determines target note with the maximum probability. If current note is selected, its length will increase. Otherwise, a new note will be generated.

(F) Generate arpeggio note. The selected note is appended to the arpeggio sequence and delivered to the RNN model as the generated notes.

Through the above process, arpeggios can be generated note-by-note along with the iterations based on the given chords. In each iteration, TCSL provides four optional notes and RNN takes the role of providing the probabilistic prediction for picking out target note with maximum probability. Meanwhile, RNN is responsible for the lengths of the generated notes. Therefore, generated arpeggios can acquire global rhythmical structure learned from the dataset since one of the strengths of RNN is the ability to model long-term dependencies.

We then conducted two sets of experiments to evaluate the output of TCSL. One set was to analyze the output of TCSL-A (see Figure-2A), given two major chord and two minor chord inputs. Another set was to compare the output of TCSL-A and TCSL-B (see Figure-2B) with different sets of operators, given the same chord inputs.

4. EVALUATION

We next evaluated the proposed algorithm from two aspects. Firstly, we evaluated the capacity of providing accurate prediction. Secondly, we evaluated the output of TCSL based on the experiments whether the arpeggios are as harmonic and various as we expected.

4.1 RNN Evaluation

In the case of major and minor datasets, we trained and validated two RNNs. In Table 1, it lists all the results concerning training folds and test folds.

<table>
<thead>
<tr>
<th>Major key</th>
<th>Train</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.804</td>
<td>0.334</td>
</tr>
<tr>
<td></td>
<td>0.747</td>
<td>0.546</td>
</tr>
<tr>
<td>Minor key</td>
<td>Train</td>
<td>Test</td>
</tr>
<tr>
<td></td>
<td>0.804</td>
<td>0.326</td>
</tr>
<tr>
<td></td>
<td>0.760</td>
<td>0.516</td>
</tr>
</tbody>
</table>

Table 1. The accuracy and log-cost of training and test sets.

Figure 4. Learning curves for training and test sets of major and minor keys.

Accuracy and log-cost are reported to indicate RNN’s performance in learning and predicting. The RNN model achieved a promising accuracy of 80.4% in training data.
and 76% in test data. From the results, it shows that minor RNN slightly outperforms the major RNN concerning test accuracy. The rational gap of accuracy between training data results and test data result demonstrates the ability of generalization of our predictive model. In Figure 4, it shows the learning curves of log-cost which describes the training and test data without over-fitting.

4.2 Algorithm-generated Arpeggios Output Evaluation

Regarding the algorithm output, the selection of 12 major keys and 12 minor keys were used in the evaluation experiment. Each key incorporated seven scale chords. We performed two experiments to evaluate our approach.

Experiment 1: In this experiment, we used TCSL-A (see Figure 2) to generate arpeggios for all scale chords with the purpose of evaluating harmony of the generated arpeggios output. Chords were imported into the model one-by-one. As a result, our method generated 144 pieces of arpeggios from the major key, and likewise the minor key. C, G (I, III scale degree of C major) and am, g (the I, VII scale degree of a minor) chords were provided for presentation. In this paper, we presented four pieces of arpeggios in the format of the piano roll (See Figure 5). Y-axis denotes the beats in time series, and X-axis indicates the pitch. The position and length interpret the pitch and duration of a note, respectively. We took 4/4 as time signature, thereby the shortest block was the eighth note, taking up half beat. Take Figure-5A as an example, starting from C1, first step to G3 with a ‘’’ symbol, down to E3 with ‘-1’, repeating E3 for two beats with 3 ‘#’, then down to G2 with ‘-2’.

Here we focused on evaluating the harmony of the presented four arpeggios. We found that the adjacent notes contain the harmonic interval ratio. The ratio of the frequencies of the pitches performs such as 1:1(unison), 2:1(octave), 3:2 (perfect fifth), 4:3 (perfect fourth), and 5:4 (major third). To most listeners, these intervals are more likely to sound consonant and well-tuned [32]. Take Figure-5A for example, G2 to C2 is a perfect fifth, E3 to C3 is a major third and G3 to G2 is an octave. Therefore, harmony is achieved in the algorithm-generated arpeggios output.

In addition to harmony, rhythm is also achieved in the generated arpeggios. We noticed that four types of note durations have appeared in the presented pieces, which took up half beat, one beat, one and a half beats and two beats, respectively. Most of the notes were eighth notes. Other longer durations interspersed among the relatively smooth rhythm to show a sense of instability and variation, which produces the pattern of rhythm.

However, the sense of harmony and rhythm of the output arpeggios may be better understood by listening to the acoustic samples. To this end, we invite readers to visit this paper's accompanying website¹ where we have aggregated a series of generated arpeggio samples on multiple chords and keys.

Experiment 2: In the second experiment, we intended to evaluate the variety of the generated arpeggios output. TCSL-B (see Figure-2B) using another set of operators was provided to compare the output of TCSL-A and TCSL-B. We produced two pieces of arpeggio on C and g chord. The results are shown in Figure 6.

When compared Figure-5A (from TCSL-A) with Figure-6A (from TCSL-B) on C chord, the results differ in three aspects. Firstly, the selections of notes are nearly different at the same beats. Secondly, the types and distributions of the intervals between two adjacent notes are mostly different, which determines the distinct fluctuations of melody contour. Further, multiple longer note durations spread out across various beats, making the definite rhythm. The above differences indicate that TCSL is capable of generating distinct arpeggios when switching to another set of operators. Because TCSL has plenty of sets of operators, we conclude that TCSL has the advantage of creating a variety of distinct arpeggios from the results of this experiment.

Likewise, we assembled the arpeggio samples generated by other sets of operators on the accompanying website. Perceptions of variety can be directly acquired from those audible arpeggios.

¹ http://www.signichat.com/suta/arp/
4.3 Discussion

Our study is dealing with the arrangement of arpeggio notes through an algorithmic approach. In comparison with the four pieces of arpeggios generated by TCSL-A, we find that the melody contours are distinct from each other. That means in our system TCSL operate without constraining the alignment of notes into a fixed template. However, the interval operators still serve as the primary production rule to softly control the arrangement of note generation.

Here we discuss the main reasons for achieving inherent harmony and rhythm pattern in our composition output. First, inherent harmony derives from the existing pieces of arpeggios which carry the harmonic interval ratios elaborated by the composers. RNN learns these harmony features and then embodied in our generations. Secondly, rhythm pattern is modeled by RNN and determined in collaboration with TCSL. Because RNN can capture the dependences on repetitive notes from the original dataset, it provides a higher probability for current note when the complete note sequence including current note is fed to RNN. Therefore, the learned harmony and rhythm will not diminish in the output of our algorithm whenever the chord inputs or sets of operators are varied. Instead, it will involve variety into the algorithmic generations.

Since our algorithm takes the advantages of generating harmonic and various arpeggios in real-time workflow, it has promising potentials to be applied in improvisation and interactive composition systems. Integrated with all kinds of sensors and HMC devices, it can capture multi-dimensional inputs and afterwards map the quantitative indicators to tonality, chord inputs, and sets of operators in our algorithm. This mapping framework exemplifies the dynamic relationship between captured user input and the generated music compositions. Therefore, our algorithm shows capability of shaping natural interactions between performers and interactive composition system and composing harmonic and various musical generations.

5. CONCLUSIONS

In this paper, we have presented a music composition algorithm for generating harmonic and various arpeggios. The algorithm incorporates an DLM, provided by RNN with GRU and a generation model supplied by TCSL. With the two core algorithmic components, inherent harmony and variety can be achieved in the generated composition. We have evaluated RNN model which indicated the efficient ability to learn global structures and to predict probabilities of the next note. The experiments on the output of algorithm revealed that in collaboration with RNN, TCSL performed a significant advantage of generating variety. Further, our research has extended the capability and potential of DLM towards the design space for interactive scenarios by addressing the gap between machine learning network and interactive systems. Users can naturally interact with the algorithmic model to manipulate musical generation simultaneously through the TUI. This paper aims at informing researchers, composers, and developers in the communities of Artificial Intelligence and Human-Computer Interaction to explore the combination of machine intelligence and human intelligence. These two parts can mutually receive reinforcement when confronting interactive systems.

The future work will firstly focus on the improvement of the current TCSL model. We intend to invite users including composers, pianists as well as inexperienced users to further evaluate the harmony and variety of the generated arpeggios. The suggestion would benefit to the enhancement of TCSL, for example, regarding the selection of operators and given chords. Besides, we will explore the scenarios, interactions, and interfaces of applying our algorithm to interactive composition systems, for instance, by shaping a multi-modal TUI which allows users to compose simultaneously like how a conductor directs the orchestra through natural gestures, gaze, together with the movement of head and arms. In the interactive scenarios, users would achieve harmonic compositions alone with continuous interactions with the interactive systems.

Acknowledgments

This work was jointly supported by National Natural Science Foundation of China (Grant 61571202) and Shanghai High Peak IV Program (Grant DA17003). The authors would like to thank Junwen Cheng for the help of the development.

6. REFERENCES


