COMPOSITION AS GAME STRATEGY: MAKING MUSIC BY PLAYING BOARD GAMES AGAINST EVOLVED ARTIFICIAL NEURAL NETWORKS

Eduardo Reck Miranda
eduardo.miranda@plymouth.ac.uk

Qijun Zhang
qijun.zhang@plymouth.ac.uk

Computer Music Research
Faculty of Technology
University of Plymouth
http://cmr.soc.plymouth.ac.uk

ABSTRACT

This paper introduces MIDI-Connect4, a program that composes music from the unfolding of a board game called Connect4. The system uses evolutionary computation to evolve from scratch a neural network that plays the Connect4 game. Music is produced when a user plays the game against the system. The system generates music by associating the moves of each player with musical forms.

1. INTRODUCTION

MIDI-Connect4 was inspired by a musical event called *Reunion*, conceived by composer John Cage, Marcel Duchamp and Teeny Duchamp in the late 1960s [2, 6]. The sounds varied according to the position of pieces on a specially prepared chessboard built by Lowell Cross. The board triggered sound-generating systems prepared by David Behrman, Gordon Mumma, David Tudor and Lowell Cross himself. Sounds were spatially distributed around a concert audience as a chess game unfolded.

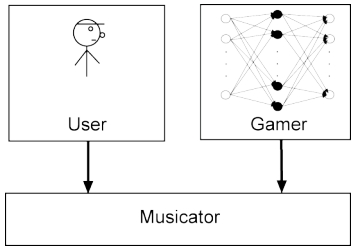


Figure 1. The architecture of MIDI-Connect4.

MIDI-Connect4 has two main components: the *gamer* and the *musicator* (Figure 1). Whereas the former is responsible for playing the game with the person interacting with the system, the latter generates music by associating the moves of each player with musical forms. The system teaches itself to play the game using neural networks and an evolutionary computation technique.

This paper is organised as follows: the following section presents the gamer. It introduces its neural network architecture followed by an explanation of its learning process. Next it presents the musicator. It introduces its method for generating music from the moves of the game. Then, it explains how the system works, followed by concluding remarks.

2. THE GAMER

The gamer features a neural network that effectively learns how to play the game. The gamer is implemented after Blondie24, a computer program by David Fogel that successfully learns by itself how to become a good player of the game checkers [3]. The architecture of the neural networks and the training process is explained below.

2.1 The Neural Networks

The architecture of the neural networks is shown in Figure 2. It is a single-layer feed-forward network: the network has one layer of connection weights and the signals in the network flow from the input units to the output units in a forward direction [4]. There are 36 input neurons, 6 output neurons and 60 hidden neurons.

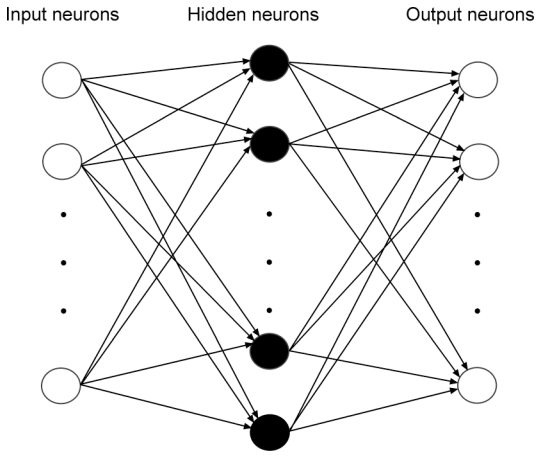


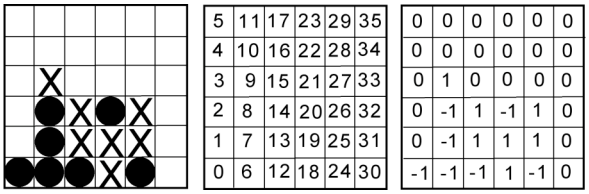
Figure 2. The architecture of the neural networks.

As MIDI-Connect4 is played on a 6×6 game "board", each lattice of the game board has an input neuron associated with it, and each output neuron corresponds to a column of the board. All input and output neurons connect to every hidden neuron. All connections between two neurons have a real number weight within the range of [0.5, 2.0] assigned to them. A threshold value is assigned to every hidden and output neuron. All these parameters are combined in a long binary string that represents the neural network. Also, it is worth noting that, although not included in its structure (i.e. not

included in the long binary string that represents it), a neural network has an important piece of information associated with it: its fitness value. The role of this information is discussed in section 2.3.

2.2 Playing the Game

Firstly, let us explain how the neural networks represent a game position on the board. An integer value of 1, -1or 0 is assigned to every input neuron according to the status of its corresponding lattice; that is whether it is occupied by the gamer, by the opponent (i.e., the person who is playing against the system) or empty (Figure 3).



5	11	17	23	29	35
4	10	16	22	28	34
3	9	15	21	27	33
2	8	14	20	26	32
1	7	13	19	25	31
0	6	12	18	24	30

0	0	0	0	0	0
0	0	0	0	0	0
0	1	0	0	0	0
0	-1	1	-1	1	0
0	-1	1	1	1	0
-1	-1	-1	1	-1	0

Figure 3. Board positions with input and output neurons.

On the left of Figure 3 is the Connect4 board with some occupied positions. Indices of input and output neurons associated with lattices on the board are shown in the figure in the middle. The figure on the right side shows the input and output values for the neurons representing the board positions that are occupied.

At this point it is necessary to shortly describe the rules for Connect4. A move means to occupy an empty square of the board. Each player makes moves in turn and a move should occupy the lowest empty square in a column. The first player that gets four connected spaces on the board (horizontally, vertically or diagonally) wins the game. The game is a draw if the board gets full without four connected spaces by either player.

How does the gamer choose a move to make? As mentioned above, it learns to play the game by itself: no game strategy or expert knowledge is given a priori. The gamer's neural networks learn the rules of the game using the Minimax algorithm. This algorithm is based on the hypothesis that a player will always make the move that will inflict the worse damage to the opponent strategy. It assumes that whereas one player (the "Max" side) will always try to maximize the value of its moves, the other player (the "Min" side) will always try to minimize the value of its moves. And the Minimax principle favors whatever move would minimize the maximum damage that the opponent could cause [3].

As an example, assume that it is the j^{th} move of the game and it is Max's turn to play. The system will list all possible moves M for Max: $M=\{ m_i \mid 1 \le i \le p\}$, where p is the number of possible moves for Max at that moment. Then Max uses the Minimax algorithm to choose the best move m_i, based on evaluations of board position. In MIDI-Connect4, where Max is a neural network gamer, the evaluation of board position is represented by the values of Max's output neurons. In order to compute output values, weight sum and activation operations (sigmoid function) are used twice. In the first time, all the inputs of the neural network multiplying the connection weights are added up to activate the hidden neurons. Then, the above-generated hidden values and related weights are computed in the same manner to obtain the output values. For Max's one possible move m_i $(1 \le i \le p)$, Min subsequently has several choices for its next move (as the game's $(j + 1)^{th}$ move), these choices are defined as a group. So there are p such groups as Max has p possible moves. In each group, Min's every possible move results in an output, and comparisons are held among them to get the minimum output $\min(i, x)$. In the i^{th} group (i.e., when Max makes move m_i), the evaluation of board position gets the minimum value when Min goes m'_x. So from 1^{st} to p^{th} group, p minimum output values $\min(i, x)$, where $(1 \le i \le p)$ are created. The Minimax algorithm then establishes the maximum of them, and finally, i indicates which move m_i is best for Max. In this way, Max avoids Min causing the worst damage. Conversely, if it is Min's turn, the Minimax algorithm will try to minimize the evaluations of board position that Max's possible moves can lead to.

2.3 Evolving Neural Network

The basic idea of Evolutionary Computation is inspired by the fact that natural evolution is such a powerful process, which has solved so many complex problems in nature. Researchers in this field believe that the solutions to difficult problems can be discovered in fascinating and unpredictable ways, by simulating the careful abstraction and interpretation of the natural process on a computer. Evolutionary computation is believed able to solve problems in planning, design, simulation and identification, control and classification [7]. Since evolution can create creatures that increase intelligence over time, it is also applied on generating machine intelligence. This is proved in the case of MIDI-Connect4, where a neural network that plays the game is evolved from scratch using evolutionary computation.

There are many evolutionary computation techniques, such as genetic algorithms, evolution strategies, and evolutionary programming. In MIDI-Connect4, genetic algorithm (GA) is used. The basic process of GA is as follows: at the very beginning, an initial population of individual structures (typically represented in bit-strings) is generated (usually randomly) and each individual is evaluated for fitness. Then some of these individuals are selected according to their performances. Next, the genetic operators, usually mutation and crossover, are applied to them, producing offspring. This loop of competition (fitness), selection, variation (genetic operations) and reproduction continues for generations, until an optimised individual is found [7]. In the following paragraph, more details about the GA's application in MIDI-Connect4 are presented.

Firstly, 200 randomly generated long bit-strings compose the initial population. One such long bit-string is a combination of thousands of real numbers using the same amount of bits. These numbers are all the parameters of a neural network, including 36*60+60*6 connection weights between neurons, together with 60+6 hidden and output neurons' threshold values. When a neural network is performing computations, this whole bit-string needs to be partitioned and the separated bit-strings are converted into real numbers. This happens when the neural networks are playing the game Connect4. That is, any pair of neural networks (e.g., neural network A and B) plays Connect4 against each other for two games, both A and B should be the first player once. Each neural network uses the Minimax algorithm explained earlier to decide its moves based on its outputs. A game finishes until either there is a winner or the board is full. If it is a draw, the fitness of player on each side remains the same, otherwise, 2 is added to the fitness of the winner, whereas 2 is subtracted from the fitness of the loser. When all the games (i.e.,199 × 200 games) in a generation finish, the fitness values of the 200 neural networks are sorted and the first 100 networks with higher fitness values will be selected. They will consist the pool of "parents" for producing the neural networks of the next generation.

In MIDI-Connect4, the genetic operators used for reproduction are crossover, mutation and clone. The neural networks taking part in the process of a genetic operation use their bit-strings directly; there is no need for conversion to real numbers. The crossover operation needs two parents: firstly, it selects a bit randomly as a starting point in the bit-strings of both parents, and then it swaps the corresponding segments, forming two new neural networks. Mutation is operated on individual neural network parents, by flipping their bit-strings at random. Finally, what clone does is to copy neural network parents without changes to the next generation. [7]. These three operations are originally set with different rates (*Rcross*, *Rmutation* and *Rclone*), which have values between 0 and 1. The sum of the rates needs to be 1. For example, if *Rcross*, *Rmutation* and *Rclone* are assigned 0.7, 0.25, 0.05, respectively, then (0, 1) is partitioned in three sections: $A(0, 0.7)$, $B(0.7, 0.95)$ and $C(0.95, 1)$. When producing a new neural network, firstly a real number x ($0 < x < 1$) is randomly generated. Then it decides which genetic operator will be used depending on where x lies in A, B or C.

Now let us see how to decide on which parents will these genetic operators be applied to in order to produce a new neural network. The algorithm used here is proportional selection. After the fitness value of every possible parent is known, a probability distribution needs to be created proportional to the fitness values. That is, if the fitness of one neural network is $Fit(i)$, where ($1 \le i \le 100$) then the probability of it being selected as a parent is calculated as follows:

$$\Pr(i) = \frac{\sum_1^i Fit(i)}{\sum_1^{100} Fit(i)} \quad (1)$$

This procedure is called the roulette wheel sampling algorithm. Each time a parent is needed, a real number e, where ($0 < e < 1$), is randomly generated, and the i^{th} neural network that has $Pr(i)$ first time larger than e (i.e., $Pr(i-1) \le e$) is chosen.

With above processes, the reproduction of a new generation of neural networks will finish when 200 new networks have been generated. Until the time of writing this paper, we have evolved the gamer with up to 200 generations of evolving neural networks, which is enough to generate reasonably good gamers.

3. THE MUSICATOR

The musicator produces music by associating the moves of the game with musical forms. Whenever a move is made on the board a short musical sequence of three notes are generated. The nature of this short sequence depends on the location that the current player decides to occupy on the board. However, the previous moves are also important as they dictate the choices available to the current player.

Let us define the set $S=\{n_1, n_2, n_3\}$ to represent the three notes for a short sequence S originated by a single move. The board of the game is associated with a two-dimensional Cartesian representation of musical intervals [5], where the value of x co-ordinate represents a note interval between n_1 and n_2, and y represents an interval between n_2 and n_3 (Figure 4). The values of x and y may be negative or positive; this is determined by the system randomly. A positive value means that the pitch of next note is higher than the previous one and vice-versa. Given a reference note n_1 the other two notes of the set are determined by the co-ordinates of the position occupied by the move. This reference note can be determined in a number of ways and it constitutes an important compositional variable; the user can specify sequences of reference notes to be used.

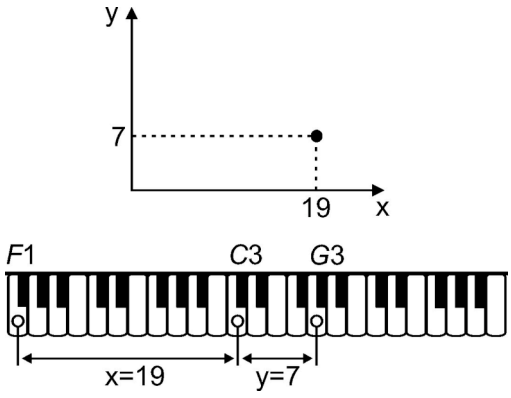


Figure 4. Cartesian representation of a short note sequence.

As a didactic example, let us fix the reference note for the first move of a game to the note middle C (i.e., C2); that is, the piece will always begin with the note C2. Suppose that the gamer made the first move: {C2, E2, F2}. Then, the n_1 for the next move by the opponent will be the same note as the n_3 of the gamer's move (in this example, F2) and so forth. The note range is set to contain three octaves, from C1 to B3. The superscripts of pieces on the game board shown in Figure 5 represent the order of moves of an hypothetical game and Figure 6 shows the representation of a sequence of moves in standard music notation. Considering that x and y can be either positive or negative, two possible resulting note sequences are given as follows:

1. {C2, E2, F2}, {F2, A#2, B2}, {B2, G2, F2}, {F2, G#2, A2},{A2, F#2, E2}, …
2. {C2, E2, F2}, {F2, C2, B1}, {B1, G1, F1}, {F1, D1, C#1}, {C#1, E1, F#1}, ….

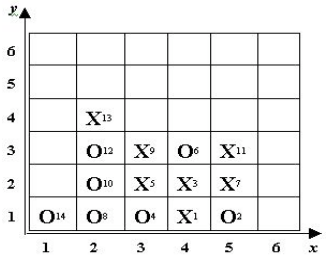


Figure 5. A sequence of moves.

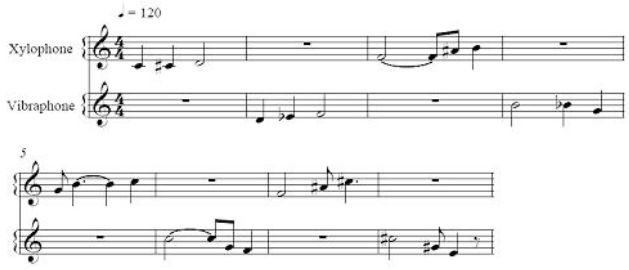


Figure 6. Excerpt of one composition resulting from the sequence of moves {C2, C#2, D2}, {D2, D#2, F2}, {F2, A#2, B2}, {B2, A#2, G2},{G2, B2, C3},{C3, G2, F2}, {F2, A#2, C#3}, {C#3, G#2, E2}.

With respect to rhythm, the durations of the notes are calculated as follows: each move generates a musical bar. In other words, each set S corresponds to a musical bar. The total duration c for a move S is fixed a priori for the entire piece. The individual durations for n_1, n_2 and n_3 are assigned randomly, under the condition that, their sum is equal to c. We also assign dynamic values (i.e., MIDI velocity values) for every note, and the values are generated randomly in the range of between 20 and 100.

4. CONCLUSION

John Cage's *Reunion* event inspired us to investigate the possibility of applying game theory to musical composition. MIDI-Connect4 is a first attempt at musical systems whose compositional rules are given by the rules of a board game. We have shown that it is possible to evolve a neural network to play the game from scratch. Although we have managed to adapt the Connect4 game to produce music we feel that the next step in this research is to devise games whose rules have closer associations with well known rules for musical compositions; e.g., harmonic progression rules. We are currently looking at the possibility of designing a musical board game based on the group-theoretic description of pitch systems proposed by Gerald Balzano [1].

The second author, Qijun Zhang, would like to thank P. W. Grant, of the Department of Computer Science, University of Wales Swansea, for fruitful discussions on evolving neural networks for playing board games.

5. REFERENCES

[1] Balzano, G. J. (1980), "The Group-theoretic Description of 12-Fold and Microtonal Pitch Systems", *Computer Music Journal*, 4(4):66-84.

[2] Cross, L. (1999), "Reunion: John Cage, Marcel Duchamp, Electronic Music and Chess", *Leonardo Music Journal*, 9:35-42.

[3] Fogel, D. B. (2002), *Blondie24 : playing at the edge of AI.* San Francisco : Morgan Kaufmann Publishers.

[4] Fausett, L. (1994), *Fundamentals of Neural Networks. Architectures, Algorithms, And Applications.* Old Tapann: Prentice Hall International.

[5] Miranda, E. R. (2003), "On the evolution of music in a society of self-taught digital creatures", *Digital Creativity*, 14(1):29-42.

[6] Revill, D. (1992), *The Roaring Silenece: John Cage – A life*. London: Bloomsbury.

[7] Back, T., Fogel, D. B. and Michalewicz, Z. (2000), *Evolutionary computation 1 Basic algorithms and operators*. Bristol: Institute of Physics Publishing.