Dorin, A., "Boolean Networks for the Generation of Rhythmic Structure", in *Proceedings ACMC 2000*, Brown & Wilding (eds), Australian Computer Music Conference, July 2000, pp38-45

Boolean Networks for the Generation of Rhythmic Structure

Alan Dorin
School of Computer Science & Software Engineering,
Monash University, Clayton, Australia 3800
aland@cs.monash.edu.au http://www.cs.monash.edu.au/~aland

Abstract

This paper describes an interactive multi-track MIDI sequencer for generating polyrhythmic patterns. Patterns are determined by the state of a set of autonomous Boolean networks, each node of which corresponds to a single musical event. This event will be triggered if the Boolean node is active and the sequencer step corresponds to its location in the pattern. Users may alter the networks which generate the patterns and change the note information they trigger whilst a sequence plays. Thus the sequencer may be used in live performance or in a studio. Additionally the sequencer may be left to generate its own patterns based on the behaviour of the Boolean networks.

Introduction

The generation of complex patterns from simple rules is a constant source of fascination for practitioners in the fields of computer generated imagery and sound [1,2,3,4,5,6,7,8]. This interest takes root in a deeper drive to comprehend the world around us. Where we may, we reduce complex outcomes to the combination of simple principles. We look for causal connections between otherwise mysterious and unpredictable events. We seek order in chaos and, in our own work, seek *complexity for free*.

This paper describes a musical event generator which has application as a multi-track interactive sequencer and generator of polyrhythms. The complex patterns it generates emerge from the neighbourhood interactions of binary switching elements. The tool is intended to work as an instrument for composition or as a stand-alone rhythm generator which, once established, runs without interference from a human user.

The combination of multiple independent rhythmic patterns gives an emergent and intricate shimmer with a "life" of its own. This phenomenon is quite irreducible to its component parts. It is this aspect of rhythm which first attracted the author to the idea of generating interwoven processes in a manner reminiscent of the functioning of a living thing.

Steve Reich hits the nail on the head when he states "Sometimes everything just comes together and suddenly you've created this wonderful organism" [9, pp7] The resulting piece of music is alive, an organism, in the sense that a multitude of interacting processes combine to produce a transient dynamic entity with an energy not apparent on the dissecting table. Michael Nyman [10, pp 149] also uses the term musical "organism" in a discussion of the work of Phillip Glass.

Xenakis' percussion work *Pleiades* [11] and Reich's pieces (for example *Drumming*, *Music for 18 Musicians* and *Sextet* [12,13,14]) are good examples of the dynamism which may emerge from the careful orchestration of simple patterns. Of course Reich is best known for his use of *phasing*- a term coined to describe the procedure of playing repetitive patterns of different lengths against one another so that they gradually shift out of (and then back into) phase. Music from African, Chinese and Cuban sources is also frequently based around complex polyrhythmic structures [15, pp 35].

Reich was, in his earlier days at least, specifically interested in the *process* by which his music arose. "Once the process is set up and loaded it runs by itself" [10, p152] The simplicity of a process such as playing a repetitious drum pattern, or the operation of mechanical tape players, contrasts strongly with the complexity of the result. Here then the composer has demonstrated a kind of power over sound which is not a reflection of their ability to conceive of intricate rhythms, so much as it illustrates an understanding of the origins of complexity in simplicity. A fascination with these ideas has played a part in the production of the software described shortly in this paper.

The behaviour of Wolfram's one-dimensional cellular automata (CA) [16] is also of relevance to the present discussion. Wolfram established a closed loop of automata which is usually depicted as a row of cells. The two end cells are assumed to be neighbours in the same way that adjacent cells within the line neighbour one another.

Wolfram's cellular automata are binary state machines–they may be in the state *on* or *off*. The state a given automaton will be in at a discrete time step immediately following the present one is determined by its present state and the present state of its two neighbours. All automata in a row are updated synchronously.

It is possible to exhaustively document the patterns Wolfram's connected machine produces by running through all the possible rules governing the state changes of its automata. Wolfram found four general types of behaviour in his machines. He found machines which:

(i) move into a homogeneous state (limit-point);
(ii) move into simple, separated, periodic structures (limit-cycle);
(iii) produce chaotic aperiodic patterns (strange attractors);
(iv) produce complex patterns of localized structures.

The relationship between these behaviours and the production of music or imagery using procedural means has been discussed in [17]. Various authors and composers have produced systems which generate musical structures from cellular automata [18, 19].

Wolfram's CA's are, like any dynamical system, capable of producing patterns for use as rhythms in music. However for various reasons cellular automata were not suitable candidates for the work presented in this paper. Instead, Boolean networks were used as the source of rhythmic complexity. The reasons for this are explained in the next sections. Following this, the present system for generating musical patterns is explained in some detail. The tool's visual interface is also described since its production required numerous decisions to be made regarding the effective presentation of a large amount of information. Finally, general thoughts are given on the utility of the system, some future work is proposed and conclusions drawn.

What are Boolean networks?

Boolean networks are a connected set of binary state machines (nodes) similar to the automata in Wolfram's CA's. Nodes in a Boolean network may be in one of two states: *on* or *off*. In this paper only *synchronous* Boolean networks will be considered. These are networks in which the node states are updated simultaneously, just as they were with Wolfram's CA's.

The future state of a node depends on the states of nodes in the network designated as that node's *inputs*. A node may feedback its own state as a self-input. Additionally, inputs may be received from outside the network. A Boolean network which receives no input from the "outside world" is an *autonomous* network. An autonomous, synchronous Boolean network is clearly a special case of the CA discussed in the previous section.

The state of a node in a Boolean network at a future time is governed by a logical rule or Boolean function which operates on the node's inputs. Examples include the NOT, AND, OR and XOR functions which have become so familiar in this digital age. The set of states a system passes through as its nodes are updated is known as its *trajectory*.

Kauffman [20, pp188] discusses the trajectories of autonomous Boolean networks with exactly two inputs to each node. The first important concept to grasp is that any actual network has a finite number of binary nodes. Therefore the machine as a whole has a finite number of possible states (specifically 2^N where N is the number of nodes). Since the network is deterministic, if it returns to a state from which it previously emerged (which it must do eventually since there are a finite number of possible states), the network must be in a *limit cycle*. The number of steps in a limit cycle may range from one (a limit point) to 2^N.

The set of machine states which lead the network to fall into a particular limit cycle is the *basin of attraction* for that limit cycle. The limit cycle is the *attractor* for the basin. A network must have at least one limit cycle but some networks have many, each with its own basin of attraction. Some basins are very broad (a large number of machine states lead to their attractor), others are quite narrow. Whilst the machine is within the basin of attraction but not yet within its limit cycle the machine is said to be in a *transient*.

Besides basins of attraction, the other major property of interest is the susceptibility of a particular attractor to disturbance. Some limit cycles are more stable than others. That is, even if some nodes in the system have their state flipped, the basin of attraction around the attractor ensures the trajectory of the system returns from its transient to the previous limit cycle. Sometimes, even major structural alterations to a network such as changes to node connections or node transition rules do not upset limit cycles.

Why use Boolean networks?

Of course multi-layered rhythms may be constructed and edited by hand in a conventional sequencer. There is no disputing the value of this technique. However the current task is the generation of such patterns using the computer as an able assistant, even a master. A system is sought which may be influenced or guided by a user but which does not require complete user-specification.

Some CA rule sets are known to be effective producers of complex patterns and cyclic activity. For example Conway's *Game of Life* falls into this category [4,5]. Some researchers have used CA's like this for music event generation [18] or generated their own CA's with desirable properties [19], however the mapping from CA to music is almost arbitrary. Why should the CA rules produce music? The researchers may as well revert to Cage's dice or coins [21, pp132] or any of a myriad of other physical systems, and then attempt an arbitrary mapping from arbitrarily chosen system to musical events– an idea this author feels was exhausted in the sixties.

Effective transition rules for cellular automata are notoriously difficult to come up with. Wolfram's studies and the studies of Sims [22] and Langton [23], as well as this author's own investigations, indicate that of the enormous range of possible rules, the majority reliably lead a system to a fixed point or short cycle, rather than to one in which lengthy cyclic or complex patterns emerge. Boolean networks reliably produce repeating patterns which may be easily altered without causing the system to fall into uninteresting limit points–although these are not completely eliminated, and nor should they be!

It is therefore primarily the properties discussed in the previous section which make autonomous Boolean networks appropriate for the present task. As mentioned, Boolean networks reliably fall into limit cycles. These may be used as simple repeating patterns to generate musical events. Transient changes may be introduced which alter a limit cycle temporarily, thereby adding variation to a pattern without removing it altogether. Alternatively, changes to the network may be introduced which produce more complex effects on the cycle. These changes may be readily carried out in real-time by a human composer as the network is running, resulting in the immediate feedback so helpful in musical composition. They may also be carried out automatically by the computer.

Changes to the transition rules of a Boolean network do not dramatically change the *kind* of behaviour of the system. This is not true for cellular automata transition rules. For example, even a minor change to the rule set of Conway's *Game of Life* may result in the total extinction of all patterned activity on the grid. Such behaviour is not (usually) desirable from a musical standpoint.

The Boolean sequencer

The Boolean sequencer of this paper presents to the user a variable number of *tracks.* Each track contains a variable number of *nodes*, each of which corresponds to a measure in a musical bar. The system works like a conventional MIDI step-sequencer in loop mode. Each track is assigned a MIDI channel and each node a pitch, velocity and duration. Tracks (and nodes) may be muted or left to sound. The sequencer regularly steps through each node in the bar playing its assigned note at the appropriate time before looping back to repeat the pattern.

Unlike the notes in a conventional sequencer, the state of each node in the Boolean sequencer is determined by the state of its neighbours in the previous bar. Each node is connected to the two nodes adjacent to it in the pattern (first and last notes are also connected to one another) and assigned a transition rule: OR; XOR; NOT or AND. When the sequencer step corresponds to a particular node, that node will only sound its note if it is in the *on* state (and the track is not muted). *Off* notes remain silent. Node states are updated according to their individual transition rules at the end of each bar.

In user-mode the sequencer's node types and note information may be altered interactively. This allows the composer to experiment with different Boolean networks and the rhythms they produce. The user interface of the Boolean sequencer is described in the following section. The system may be left to run its own course in an automatic mode which relies on random perturbation of the Boolean networks to move the system through complex wandering trajectories.

The Boolean sequencer's U.I.

This section focuses on the visual design of the sequencer interface. Users may interact with the interface using the computer keyboard and mouse. In future versions MIDI keyboard operation will be incorporated also.

It was decided that a traditional piano-roll interface such as that found on most off-the-shelf-software would not be suitable for the Boolean sequencer. Reasons for this included the need to:

(i) represent node types (XOR, AND etc.);
(ii) have the sequencer loop rapidly across a single bar whilst it is playing (This accentuates the need to clearly indicate which note is sounding at any given time, allowing the user to distinguish a sound from many others and identify its visual representation);
(iii) indicate whether a given node was sounded/will sound during the current bar.

In addition, some conventional interface requirements were relevant to the Boolean sequencer including the need to:

(i) represent pitch, velocity and duration;
(ii) represent node temporal ordering;
(iii) allow simple editing of node parameters during sequence playback;
(iv) represent the current location in the bar;
(v) represent multiple tracks simultaneously.

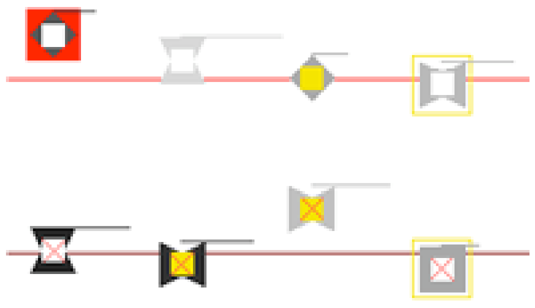

Fig. 1: Snapshot of the Boolean sequencer interface

Figure 1 shows two sequencer tracks of four nodes each. The (top) track indicated by the bright red center-line is currently selected for editing. The (top left) node indicated by the solid red square is the currently selected node. Hollow yellow squares indicate the nodes (right most) in the sequence which are currently being triggered. The rule determining the behaviour of a node (its type) is indicated by the icon used to represent it. There are presently four node types (fig. 2).

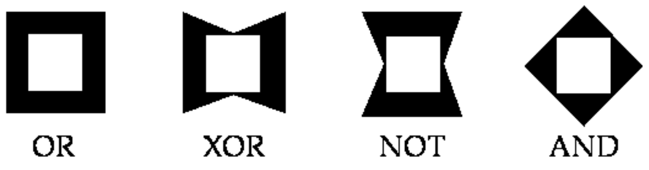

Fig. 2: Sequencer iconography

--[An unscientific aside: The "logic" for developing these icons came from considering a node's behaviour in the context of the sequencer. Whilst the reasoning requires a little creativity, once learned the node representations are (hopefully) easy to remember…

A NOT node switches on if neither of its neighbours is on at the previous time step. Hence its sides are hollow to indicate that the adjacent nodes were not on. An XOR node requires either of its neighbours to be on at the previous time step (its sides are flat) but not both neighbours (its top and bottom are hollow). An OR node requires either of its neighbours to be on at the previous time step (its sides are flat) but it is not fussed if both are on (its top and bottom are flat). The AND node requires *both* of its neighbours to be on (its sides, top and bottom actively poke out from the icon's centre).

Hopefully the reader can find some sense in this description. If not, he/she will just have to learn the node icon shapes by heart to use the sequencer effectively. Thus ends the unscientific aside.]--

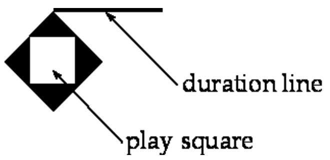

Fig. 3: Node components

The horizontal lines from the node tops (fig. 1&3) indicate a note's duration. The grey-level of a node indicates its MIDI velocity ranging from black (velocity=0) to nearly white (velocity=127). A track which has been muted will display with a red X drawn in the *play square* of each node. The play square is drawn in yellow when a node is in the *on* state, otherwise it is left vacant.

The horizontal line through each track indicates middle C. The positions of nodes above and below this indicate their relative pitch. Exact pitches are not visualized–this sequencer was intended primarily for users who *listen*.

As mentioned earlier, the yellow square in each track surrounds the currently triggered step in the sequence. The square shifts up and down with the current node pitch as the sequence progresses. This method of indication was used instead of the vertical bar found in many sequencers because a vertical bar (which travels horizontally as the sequence progresses) gives no visual indication of the musical event which it triggers. By having the indicator move up and down with pitch and horizontally with time, a visual indication of the pattern's flow is given.

Comments on using the sequencer, its present and future interface

The number of tracks, number of nodes per track and the playback speed of the sequencer can all be determined by the user. In the current system they cannot be varied after initiation but it is anticipated this feature will be incorporated eventually.

Although it is possible to operate the interface with a mouse, the computer keyboard shortcuts provided to alter the sequencer's parameters are simpler to activate during playback. All aspects of nodes and tracks may be altered via the appropriate keyboard commands in real-time.

Presently the pitch, duration and velocity of a node may be altered incrementally up or down using p/P, d/D and v/V shortcuts. This is clearly an ineffective means of specifying note parameters. Instead, these values might be read directly from a MIDI keyboard as a user plays a note. Nevertheless, to date MIDI keyboard specification of note data has not been required and the feature is presently unimplemented.

At present all sequencer tracks are updated synchronously. The system therefore runs like a conventional sequencer or drum machine in this regard. It is hoped in future that each track will have its own speed parameter so that patterns may shift in and out of phase with one another in the fashion of the rhythms used by Reich.

A further anticipated addition to the system is the incorporation of an arpeggiator mode. Nodes in a sequence might be taken as commands to raise or lower a note (or notes) in a complex pattern determined by the Boolean network.

Extensive experimentation with the present system has revealed it to be a versatile tool for rhythm construction. Whilst this author finds careful tweaking to be needed, especially of note pitches, other composers might find pleasing patterns form almost at random.

With percussive sounds the sequencer excels at complex rhythm production and is easily tweaked to maintain interest through variation. Complex developing sounds and samples may also be triggered by the sequencer. As these are truncated, triggered and re-triggered in unusual patterns complex timbral/rhythmic effects emerge.

This author has found the use of the NOT and AND node types alone to be sufficient to generate a diversity of multi-bar patterns. An individual track/network regularly falls into limit cycles of two to four bars using only these node types. The addition of node types which distinguish between left and right neighbours is also being considered to produce patterns which trigger events that move left and right through the bar.

Bar lengths of less than five nodes/beats per network seem to fall into very short and uninteresting limit cycles. More interesting results may be achieved using networks with more nodes than this although beyond about ten nodes per network there is no noticeable increase in the complexity of pattern produced. In the future it would be interesting to document the behaviour of networks with respect to the number of nodes they contain.

Implementation

The Boolean sequencer was implemented on a Macintosh G3 running MacOS 8.5. It was written in C++ using Metrowerks Code Warrior, the OpenGL library for Macintosh v1.1, and Opcode's OMS 2.3.8. OMS and OpenGL need to be installed to run the sequencer.

Conclusion

Like any art-making tool, one view is that it is inevitably up to the artist to make the most of its capabilities. Like any algorithmic means of art production, its output may be taken "as is" to represent the process which underlies it. In this case the result may be worthy in and of itself, or simply irrelevant. Take your pick!

Whichever view you hold, Boolean networks have been shown to be a viable means of producing complex patterns which are nevertheless easily manipulated by a user. These patterns may be usefully employed in the production of complex, changing rhythmic structure.

Bibliography

1 Prusinkiewicz, P., Lindenmeyer, A., "The Algorithmic Beauty Of Plants", Springer Verlag, 1990

2 Turk, G., "Generating Textures on Arbitrary Surfaces Using Reaction - Diffusion", SIGGRAPH 91, Computer Graphics, Vol. 25 No. 4, July 1991, ACM Press, pp289-298

3 Turing, A.M., "The Chemical Basis of Morphogenesis", Philosophical Transactions of the Royal Society, London, B, Vol. 237, 1952, pp37-72

4 Gardner, M. "Mathematical Games: The Fantastic Combinations of John Conway's New Solitaire Game 'Life'", *Scientific American*, 1970, 223 (4), pp120-123

5 Gardner, M. "Mathematical Games: On Cellular Automata, Self-Reproduction, the Garden of Eden and the Game 'Life'", *Scientific American*, 1971, 224 (2), pp112-117

6 Dahlstedt, P., Nordahl, M. "Living Melodies: Co-evolution of Sonic Communication", in Proceedings, *First Iteration*, Dorin & McCormack (eds), CEMA, Monash University, Australia, 1999, pp56-67

7 Hiller, L., "Composing with Computers: A Progress Report", Computer Music Journal, Vol. 5, No. 4, 1981

8 Laske, O., "Composition Theory in Koenig's Project One and Project Two", Computer Music Journal, Vol. 5, No. 4, 1981

9 Schwarz, K.R., "Music For 18 Musicians, Revisited", liner notes in CD, *Music For 18 Musicians*, Reich, S., Nonesuch Records/Warner Music, 1997

10 Nyman, M., "Experimental Music, Cage and Beyond", 2nd edition, Cambridge University Press, 1999

11 Xenakis, I., "Pleiades", Grammofon AB BIS, 1990

12 Reich, S., "Drumming", Elektra / Nonesuch Records, 1987

13 Reich, S., "Music For 18 Musicians", Nonesuch Records, 1997

14 Reich, S., "Sextet . Six Marimbas", Elektra / Asylum / Nonesuch Records, 1986

15 Copland, A., "What To Listen For In Music", first published, McGraw-Hill, 1939, new edition, Mentor Books, 1999

16 Wolfram, S., "Universality and Complexity in Cellular Automata", in *Physica 10D*, North-Holland, 1984, pp1-35

17 Dorin, A., "Classification of Physical Processes for Virtual-Kinetic Art", in Proceedings, *First Iteration*, Dorin & McCormack (eds), CEMA, Monash University, Australia, 1999, pp68-79

18 McAlpine, K., Miranda, E., Hoggar, S.,"Making Music With Algorithms: A Case-Study System", Computer Music Journal, Vol. 23, No. 2, 1999, MIT Press, pp9-30

19 Dorin, A., "Liquiprism", in *Process Philosophies*, curated by Dorin & McCormack, First Iteration Conference, Monash University, Australia, 1999.

20 Kauffman, S.A., "The Origins of Order - Self Organization and Selection in Evolution", Oxford University Press, 1993

21 Ford, A., "Illegal Harmonies, Music in the 20th Century", Hale & Iremonger, 1997

22 Sims, K., "Interactive Evolution of Dynamical Systems", *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference of Artificial Life*, Varela & Bourgine (eds), MIT Press, 1992, pp171-178

23 Langton, C.G., "Studying Artificial Life with Cellular Automata", Physica 22D, North-Holland, 1986, pp120-149