Nodewebba: Software for Composing with Networked Iterated Maps

Bret Battey
Music, Technology and Innovation Research Centre
De Montfort University
bbattey@dmu.ac.uk

ABSTRACT

Nodewebba *software provides a GUI implementation of the author's "Variable-Coupled Map Networks" (VCMN) approach to algorithmic composition. A VCMN node consists of a simple iterated map, timing controls, and controls for mapping the node output to musical parameters. These nodes can be networked, routing the outputs of nodes to control the variables of other nodes. This can enable complex emergent patterning and provides a powerful tool for creating musical materials that exhibit interrelated parts.* Nodewebba *also provides API hooks for programmers to expand its functionality. The author discusses the design and features of* Nodewebba, *some of the technical implementation issues, and a brief example of its application to a compositional project.*

1. INTRODUCTION

The use of pseudo-random number generators is foundational to many classical algorithmic music techniques. One well-established approach for generating pseudo-random numbers is to use Lehmer's Linear Congruence formula (LLCF) [1], an iterative map:

$$x_t = (x_{t-1}a + b) \bmod m \qquad (1)$$

The variables are optimized to provide a maximal possible period of non-repetition for a given computer's architecture. LLC falls short of uniform randomness in the short term [1], and in fact tuples of successive values exhibit a type of lattice structure [2]. Of course, pure uniformity isn't a necessity for most algorithmic-music applications. In fact, if one significantly deoptimizes its variables, LLCF can become a useful and flexible pattern generator, exhibiting a range of behaviors from decay or rise to steady state, simple periodicity, layered periodicity or unpredictable self-similarity. In a previous paper [3], I analyzed this range of behaviors and provided some guidelines to working with its parameter space. Later work by Warren Burt demonstrated an approach to LLCF that utilized a broader range of *a* and *b* variables to enable compositional exploration of varying degrees of "almost random" behavior [4]. Recent audiovisual artwork by Francesc Pérez utilizes LLCFs to control video and audio granulation processes [5,6].

The primary purpose of my aforementioned paper was to introduce the concept of Variable-Coupled Map Networks (VCMNs). A VCMN consists of a set of interlinked nodes. The core of each node is an iterated map function. The output of any one of these nodes may set a function variable in itself or any of the other nodes. Each node has a wait-time between iterations, which can also be set or controlled by other nodes. The node outputs can then be mapped to musical parameters. In theory, a node can be any iterative map, but my research focused solely on LLCF, due to its simplicity of implementation, fixed output range and small number of variables.

VCMN configurations, particularly those including feedback mechanisms, can readily exhibit "emergence", where "properties at a certain level of organization… cannot be predicted from the properties found at the lower levels." [7] Further, the paper demonstrated counterpoint behaviors between the nodes could arise when those nodes were mapped to multiple note streams or note parameters. Also notable was the capacity of VCMNs to make coherent rhythmic gestures even with entirely non-quantized timing values.

VCMNs had the disadvantage of still requiring coding to implement, making it time-consuming to configure and alter networks and explore their musical potentials. Nonetheless, I used the approach to generate some crucial materials in several compositions in the 1990s and 2000s. It was only with my 2011 audiovisual work *Clonal Colonies*[1] for the Avian Orchestra that I used VCMN as a primary tool in the creation of a large-scale work. The code I developed for and insights derived from composing the work inspired the creation of *Nodewebba*. An open-source project developed in the Max 7[2] programming language, it provides a GUI-based environment for using LLC and VCMN for pattern generation for music and other media. It provides both MIDI and floating-point outputs for mapping to a variety of targets, and it readily allows programmers to add additional functionality and integrate *Nodewebba* with other projects.

[1] Available at http://BatHatMedia.com/Gallery/clonal.html
[2] https://cycling74.com

2. FEATURES AND INTERFACE

2.1 Node Interface

Nodewebba provides six nodes, each with five parameters that can optionally be controlled by other nodes: *a* variable, *b* variable, rhythm, duration, and velocity. Each node has a GUI interface such as shown in Figure 1, supporting configuration of the LLCF, the mapping of incoming data to node parameters, and routing of MIDI output.

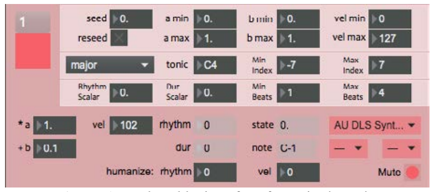
1
seed 0. | a min 0. | b min 0. | vel min 0
reseed | a max 1. | b max 1. | vel max 127
major | tonic C4 | Min Index -7 | Max Index 7
Rhythm Scalar 0. | Dur Scalar 0. | Min Beats 1 | Max Beats 4
*a 1. | vel 102 | rhythm 0 | state 0. | AU DLS Synt...
+b 0.1 | dur 0 | note C-1 | — | —
humanize: rhythm 0 | vel 0 | Mute

Figure 1. *Nodewebba* interface for a single node.

The node design supports a number of functions not addressed in the original VMCN article. For example, a "reseed" option can be toggled, so that when a node is stopped and restarted, it will re-initialize the node with the given seed value, rather than using the last state of the node. This can enable more repetitive behavior.

While LLCF is normally discussed and analyzed on the basis of its state *x* being an integer, my implementations have set *m* to 1.0, with *x* being float-point. Knowing that all state values are in the $0 \le x < 1$ range facilitates mapping of values in the system. New with *Nodewebba* is the ability to use negative values for *a* and *b*, which can provide useful variation in available pattern types — such as an inversion of the characteristic upward curve. Also new in a node is the ability to designate a minimum and maximum value for the *a* and *b* variables. If another node's output is routed to one of these inputs, in the incoming data is mapped to the given range. It is in allowing $|a|$ in particular to be greater than 1 that more complex patterning from a node, particularly unpredictable self-similarity, is enabled.

For MIDI note output, a node can designate a musical mode, the tonic for that mode, and then a minimum and maximum index into the mode. The LLCF state-variable is then mapped to the index range. For example, in Figure 1, the state-variable is 0, which mapped to the index range of -7 to 7 will yield a -7. For a major scale with a tonic of C4, the -7 will yield a C3.

For MIDI-note output, a node can be assigned a musical mode, the tonic for that mode, and then a minimum and maximum index into the mode. The LLCF state is then mapped to the index range. For example, in Figure 1, the state is 0, which mapped to the index range of -7 to 7 will yield a -7. For a major scale with a tonic of C4, the -7 will yield a C3. While conceptually convenient for many purposes, one shortcoming of this approach is that it takes more effort to figure out the correct configuration needed to keep note output between certain values — such as range limits of target instruments. If a user wants to map to tunings, modes, or musical parameters other than those provided — or other media targets altogether — the floating-point state-value output of a node can accessed directly and mapped via an API (see 3.3 below).

Rhythm and duration are determined by scalars that are mapped to minimum and maximum integer beat values. Though "beat" was chosen as a user-friendly term, this really refers to 16th-note ticks of a master clock controlled by the global tempo setting. The scalars can either be hand-specified through the interface or driven by another node. The duration scalar can exceed 1.0, creating notes longer than the maximum rhythm value.

So quantized rhythm is foundational to *Nodewebba*. This unfortunately does preclude using *Nodewebba* to create some of the interesting non-quantized rhythmic effects that VCMNs can produce. Tuplet relationships can be established between nodes, particularly if one configures each node to have only a single rhythmic value. On the other hand, variable-length tuplet notes that form neatly aligned groupings cannot always be ensured.

Indicating a randomness range value can humanize the start time and velocity of the MIDI data. The floating-point output is not humanized, since this would make it impossible to ensure such consistent, repeatable behavior in most network configurations where rhythm is determined by node outputs.

Finally, the user can indicate a target MIDI device, channel, and patch number via menus. A "Mute" toggle allows one to let the node continue its activity while muting its MIDI output. In this case, the state-variable output continues, so the node can continue to operate as part of the network even while silent.

2.2 Matrix Interface

A matrix interface (Figure 2) allows easy, realtime configuration of the network, with source node-outputs on the top and target node-inputs on the left. Push buttons allow the matrix to be cleared or randomly populated. Presets can be stored, and sets of presets can be saved to or retrieved from disk.

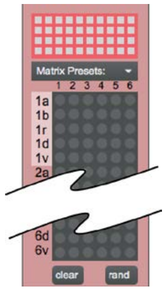

Figure 2. *Nodewebba* matrix interface.

2.3 Transport and MIDI Control

A main transport interface provides a main on/off function, external sync on/off, global tempo, and preset defining, storage and retrieval for the transport and node configuration. A MIDI Control window allows the user to assign MIDI controller inputs to the transport functions as well as to key parameters for each node: on/off; mute; reseed; a min/max; b min/max; velocity min/max.

3. IMPLEMENTATION ISSUES

3.1 Initialization and State Logic

System design solutions that support intuitive and consistent results for composers are less obvious than it might seem at first consideration. This can be demonstrated with a few example scenarios, describing the solutions established with *Nodewebba* version 0.05.

Consider a single node, seed 0, *a*=1, *b*=0.1. When starting this node, a logical expectation is that the first emitted state will be 0, not 0.1 Further, if the user later changes the seed, the next state output should be this new seed. Thus, the core iterated map enters a ‘seed changed’ state when the user provides a new seed. When fired, it emits this seed and then enters the ‘iterating’ state, where further firing results in iteration of the map.

Consider two nodes, N1 and N2. N1 controls the *a* and *b* variables of N2, and visa versa. For this to work in a consistent fashion, the firing of all nodes must occur prior to updating the nodes with the new emitted control values. Otherwise, for example, the N1 might iterate, emit its new state, and change N2’s variables before N2 iterates.

To ensure repeatable results, rhythm is implemented through a clock pulse at a 16th-note rate in the given tempo. At each clock pulse, all nodes first push the last received control data into an active state. Then all nodes increment a counter. Once the counter receives enough triggers to reach the rhythm beat-count, the node fires: the iterated map is activated and its resulting state-value is emitted and sent to any targets node inputs in the network as potential future control data.

In the case of a node controlling its own rhythm and duration, the user would likely expect rhythm and duration to directly reflect the emitted state of the node. That is, if the node emits a 0, the shortest note value should ensue at that time, rather than on the next firing. Likewise, if the node emits a 1, the longer possible note value should ensure. To support this behavior, the firing step places the node in the ‘ready to emit MIDI’ state. After all nodes have been invited to fire, they are all then invited to emit MIDI data. If a node it ready to emit, it will calculate rhythm and duration based on most recently received control data (which might come from nodes that just fired ‘now’), generate the MIDI output and set the beat counter target for next firing of the node.

In summary, then, at each metro clock *Nodewebba* executes ‘update inputs’ for every node to gather the latest variable-control inputs, then calls each node to update its counter and fire if ready, and then has each fired node update its rhythm and duration based on any just received control data and then emit the MIDI data.

3.2 Other Issues

Allowing negative values for *a* and *b* means that a standard modulo function no longer suffices. Instead, the function must wrap when it receives a negative number. This is implemented as follows:

$$\begin{aligned} &\text{for } x \geq 0,\ f(x) = x \bmod 1 \\ &\text{for } x < 0,\ f(x) = \big(1 - (x \bmod 1)\big) \bmod 1 \qquad (2) \end{aligned}$$

The outer mod 1 in the latter formula ensures that a 1 returned by the inner mod becomes 0, thereby maintaining the expected $0 \leq x < 1$ output range. In Max, implementation is complicated by the fact that Max can return a negative zero in floating-point calculations, which will return *true* if tested to see if it is less than zero. The current solution tests instead to see if *x* is less than 0.000001.

3.3 Postprocessing and API

Though *Nodewebba* can be used as-is, many composers might wish to provide additional post processing, automation or links to other systems. This is particularly true given that no musical knowledge is embedded in the network itself, and only minimal musical knowledge (in the form of modes) is supported in the mapping. Therefore, additional logic based on specific musical intents may be required.

Post processing is implemented in a subpatcher that receives the output messages from the nodes. The MIDI notes generation and humanization already occur here. Working with the source version of Nodwebba (versus the standalone), a Max coder can easily add additional post processing functionality here without having to intercede with the lower-level node code.

Further, API hooks are provided in the form of accessible variables (Max sends and receives) and a standardized naming scheme that includes node numbers in the variable names. For example, one can control the on/off state of Node 5 by addressing a [send 5o] object, or receive the state-variable output of Node 3 with [receive 3val]. Thus a coder can interface with *Nodewebba* without needing to touch any *Nodewebba* code.

3.4 Synchronization

A Max *hostsync~* object allows a ReWire-enabled external sequencer to control the *Nodewebba* transport and clock via the ReWire protocol, preventing timing from slipping between the two programs when recording *Nodewebba* output to the sequencer.

4. APPLICATION

Extensive details regarding the use of VCMN in the composing of *Clonal Colonies* can be found elsewhere [8]. Here we will just look at a few examples of external control and post processing in that piece. In short, each instrument (flute, bass clarinet, violin, cello, piano, and gongs) was assigned to one node, since this was conceptually the simplest way to approach the work. Nonetheless, as noted in the original article, the systems still readily become too complex to treat analytically: one must use exploration and heuristics to develop compositional possibilities.

One original impulse behind the VCMN technique was the idea that some network configurations might exhibit higher-order emergent “change in the nature of change”, leading to a greater sense of dramatic plateaus or even trajectories. Instead, complex networks with many feedback links tend to create “too much variety”, where the output risks being perceived as exhibiting continuous change without significant-seeming patterning. In these cases, too, it can be very hard to find meaningful ways to intercede with the settings to provide a sense of musically coherent transformation of behavior. In practice, overcoming this requires explicit design or external control mechanisms, or by placing some nodes in a position of clear hierarchical control over the system, typically operating at slow rates of change.

In *Clonal Colonies*, to address the wide range of potential network configurations and the lack of predictability in results, I connected an external MIDI slider-controller box to numerous parameters of the system, allowing relatively fast exploration of behaviors and gradual development of a structured improvisation.

Post processing routines were also central to addressing aesthetic and practical technical issues. One issue with VCMN is lack of a natural phrasing mechanism. Besides the resulting risk of monotony, the lack of phrasing can be particularly problematic when writing for wind instruments, since the performer needs opportunities to breathe. Inspired by practices common in Hindustani classical music, I decided that certain notes in the mode would receive extra emphasis — in this case by trilling them and using them to end phrases. Post-processing code detected when a node generated these pitches. The code then sent the instructions to generate a trill and turned off the node. The node would then wait for a given duration — set via the external controller — before turning on again. Thus this pause-duration control essentially served as a core density control for the whole ensemble, providing an important tool for high-level shaping of dramatic form, not provided in the VCMN itself.

Several takes were then captured to a sequencer prior to editing. This served as a sketch for the work. I gave myself the creative constraint of making the captured output of the system more convincing, not through editing, but by changing its context through the addition of computer rendered sound. In this sense, I believe VCMN and *Nodewebba* may often serve well as generators of ideas and rough structures on which a composer can then elaborate, gaining creatively from the surprises generated from the system.

5. CONCLUSIONS

Nodewebba makes it considerably easier for composers to experiment with and apply both simple LLC approaches and more advanced VCMN techniques. It also facilitates live performance with VCMN. It does not significantly add to the originally proposed VCMN concept, but its development did lead to clarification of some aspects of system design for consistent and intuitive usage.

It would be ideal for *Nodewebba* to allow non-quantized rhythmic output, but this would require creation of mechanism that would allow this while also providing strict repeatability. Clearly, too, there is room to explore iterative functions other than LLCF. An ideal implementation of *Nodewebba* would allow selection of different functions. Functions that required different variable counts or don’t have a fixed output range, however, would offer a significant design challenge.

Nodewebba binaries and video demonstrations are available for download at http://BatHatMedia.com/Software/Nodewebba, and the source is available at https://github.com/bbattey/NodeWebba.

6. REFERENCES

[1] C. Ames, “A Catalog of Sequence Generators: Accounting for Proximity, Pattern, Exclusion, Balance and/or Randomness,” *Leonardo Music J.*, vol. 2, no. 1, pp. 55–72, 1992.

[2] P. L’Ecuyer and F. Blouin, “Linear congruential generators of order K>1,” *Proc. 1988 Winter Simul. Conf.*, pp. 432–439, 1988.

[3] B. Battey, “Musical pattern generation with variable-coupled iterated map networks,” *Organised Sound*, vol. 9, no. 2, pp. 137–150, 2004.

[4] W. Burt, “Algorithms, microtonality, performance: eleven musical compositions,” PhD Thesis, University of Wollongong, 2007.

[5] F. M. Pérez, “Tècniques de microsampling amb generadors de congruències lineals,” 2013. [Online]. Available: http://openaccess.uoc.edu/webapps/o2/bitstream/10609/19073/6/martifrancescTFM0113memoria.pdf [Accessed: 25-Feb-2016]

[6] F. Peréz, “Speech 2 [video],” 2015. [Online]. Available: https://vimeo.com/119713106. [Accessed: 12-Feb-2016].

[7] C. Emmeche, S. Køppe, and F. Stjernfelt, “Explaining Emergence: Towards an Ontology of Levels,” in *Systems Thinking, Volume 1.*, G. Midgley, Ed. London: Sage, 2003.

[8] B. Battey, “Creative Computing and the Generative Artist,” *Int. J. Creat. Comput.*, vol. 2, no. 1, 2016.