Generative Composition with *Nodal*

Jon McCormack, Peter McIlwain, Aidan Lane, and Alan Dorin

Centre for Electronic Media Art
Monash University, Clayton 3800, Australia
Jon.McCormack@infotech.monash.edu.au
www.csse.monash.edu.au/∼jonmc

Abstract. This paper describes a new generative software system for music composition. A number of state-based, musical agents traverse a user-created graph. The graph consists of nodes (representing events), connected by edges, with the time between events determined by the physical length of the connecting edge. As the agents encounter nodes they generate musical data. Different node types control the selection of output edges, providing sequential, parallel or random output from a given node. The system deftly balances composer control with the facilitation of complex, emergent compositional structures, difficult to achieve using conventional notation software.

1 Introduction

The goal of any Artificial Life (AL) or generative composition system should be to offer possibilities and results unattainable with other methods. A number of authors have suggested that the *emergence* of novel and appropriate macro behaviours and phenomena — arising through the interaction of micro components specified in the system — is the key to achieving this goal [1–3]. While simple emergence has been demonstrated in a number of AL systems, in the case of musical composition, many systems restrict the ability to control and direct the structure of the composition, conceding instead to the establishment of emergence as the primary goal.

This focus on emergence exclusively, while interesting in terms of the emergent phenomena themselves, has been at the expense of more useful software systems *for composition itself*. The aim of the work described in this paper is to design and build a generative composition tool that exhibits complex emergent behaviour, but at the same time offers the composer the ability to structure and control processes in a compositional sense. The idea being that the composer works intuitively in a synergetic relationship with the software, achieved through a unique visual mapping between process construction and compositional representation.

This paper describes a new kind of generative music composition system, which we call *Nodal* (Fig. 1). The system uses spatial, directed graphs that are traversed in real-time by one or more state-based agents, known as *players*. The players traverse the graphs moving along edges and responding to state changes

specified in vertices. The time taken for a player to move along an edge is proportional to its length. The composer designs and constructs the graph visually, using the mouse to create nodes and connect them on a two-dimensional surface. The visual representation of both structure *and* process provides a direct mapping to the emergent musical output. A threaded architecture permits real-time editing and visualisation while the network is being played, allowing modifications to the network as part of a performance if necessary. This real-time design aspect assists the composer in quickly forming intuitive relationships between visual structure and musical results.

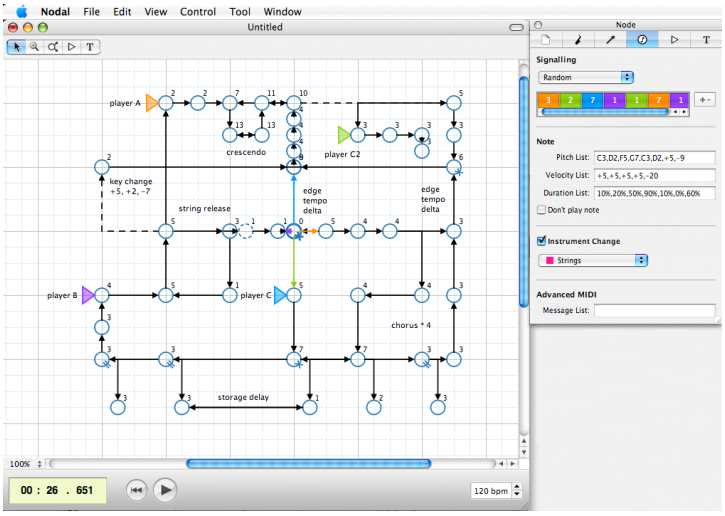

Fig. 1. Screen shot of *Nodal* showing the network editor (left) and node editor (right).

1.1 Related Work

While use of graphs for musical analysis has received some attention (e.g. [4]), using graphs as generative compositional tools is less well studied in the literature. There are a number of similarities in this approach to previous generative musical specification systems, such as generative grammars, finite state automata, Petri nets and Predicate Networks [5–8]).

The 'ant music' system of Guéret et. al. used a connected graph structure with nodes representing MIDI events [9]. Ants traverse the network, leaving pheromone trails on the edges. Pheromone intensity acts as transition weight and pheromones naturally evaporate to avoid stasis. The ants begin at randomly assigned vertices and choose to move along edges based on the amount of pheromone present on that edge.

The *Luminaria* component of the game software *Electroplankton* developed by Japanese artist Toshio Iwai, used a fixed, regular grid of points traversed by a number of musical agents. Users of the game manipulate arrows at each grid point allowing them to point to one of the eight nearest neighbours, thus controlling the movement of the agents as the move from one grid point to the next. Each grid point is mapped to a different note, so changing the agent's path changes the melody produced.

2 Nodal Operation

In this section the basic structure and operation of the *Nodal* system is described. The main user interface is shown in Fig. 1. Users click in the *network editor* window to create nodes, and drag to create connections (edges) between nodes. Alternatively, nodes may be entered by playing any MIDI keyboard — the node is created under the current mouse position with state data (pitch, volume, channel, etc.) recorded from the incoming MIDI information.

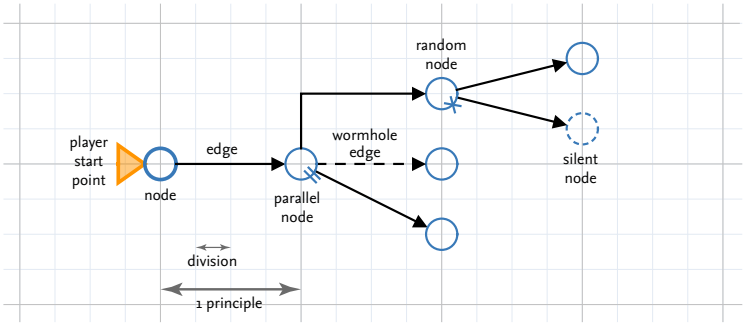

Fig. 2. Elements of a network: player start point, nodes, edges and the timing grid. The darker grid lines represent principles, lighter grid lines are divisions. In this example there are four divisions per principle.

2.1 Space and Time

Music can be considered the organisation of sounds in space and time. *Nodal*'s interface literally reflects this consideration. Creating a composition involves interactive placement of nodes in space and the connection of nodes by edges. As the spatial distance between nodes represents the musical time between events, accurate placement is crucial. The editing window uses a grid-based system to achieve this (Fig. 2). Nodes are interactively placed and snap to the nearest grid point.

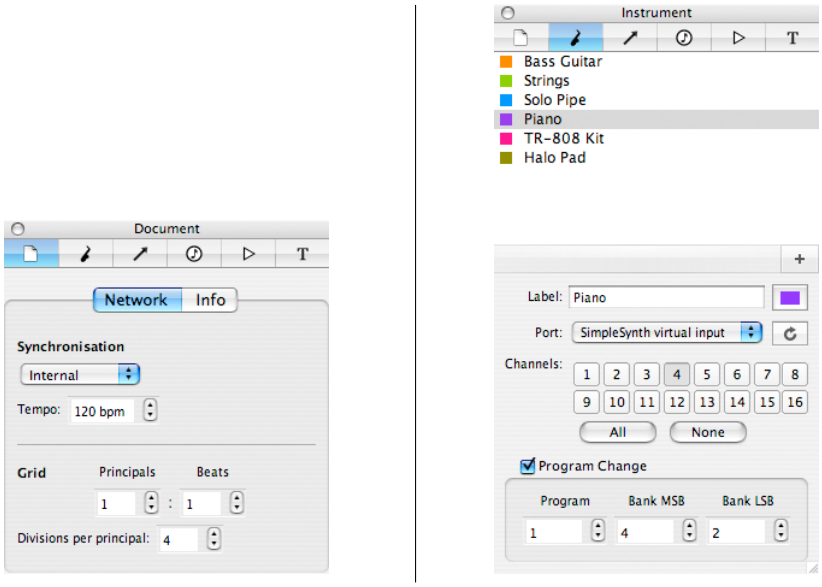

Fig. 3. Time control in *Nodal* (left) and instrument editing (right).

Space (hence time) is divided horizontally and vertically into *principles*. A principle represents a number of *beats* and the overall tempo of traversal specified beats per minute (bpm). Changing the tempo or beats to principles ratio changes the overall timing that agents use to traverse the network. The time taken to move a distance of d principles is given by the equation:

$$t = d\left(\frac{b}{p\tau}\right) \tag{1}$$

where b/p is the beats per principle ratio and τ the tempo in beats per minute.

To assist with developing standard time signatures and metrical structures, each principle is further subdivided into a number of *divisions*. Changing the divisions per principle does not affect the timing information, it allows placement of nodes with appropriate timing within the principles. Fig. 3 shows the document editing window in *Nodal*'s user interface, responsible for user-editing of these parameters. The purpose of divisions is to permit accurate construction of networks with common musical timing (half notes, quarter notes, dotted notes, triplets, etc.). For example, assuming one principle per four beats and four divisions per principle makes each division one quarter note. One can change the divisions per principle to eight and then create eight notes, without changing the timing.

Creatively, there has been some criticism of the use of rigid Cartesian structures such as grids [10, 11], which tend to enforce restrictions on musical possibilities by their constraints. In response, we emphasise two points: firstly, without a

system for time quantization, metrical timing structures would not be possible; secondly, it is easy to turn the grid feature off and explore other geometry in relation to timing. Fig. 10 shows an example of this where node distribution is based on non-linear ratios, such as logarithmic spirals and galaxy formation simulation (node position mapping to stars).

2.2 Players

A *player* is a musical agent that traverses the graph in real-time. Players play with one or more *instruments*, which represent MIDI output channel(s) and program change messages (Fig. 3). A player may change instruments as it traverses the network.

Players contain a *state*, consisting of a current pitch, velocity, duration and instrument. As the player traverses the network its state is updated by the nodes it encounters.

2.3 Nodes and Edges

Vertices of the graph are referred to as *nodes* and displayed as a blue circle shape in the network editor. For a player traversing the network, the normal action when it encounters a node is to play a note. A node contains state modifying information, which consists of:

- a list of pitch change information;
- a list of note-on velocity information;
- a list of note duration information;
- instrument change (instructs the player to change instruments);
- the option to update player state information without playing a note;
- a general list of MIDI instructions to play.

Player state changes can be absolute or relative. For example, a pitch value of +2 increases the player's current pitch by two semitones, G4 sets the player's pitch to the specified note. Duration may be relative, absolute (measured in beats) or as a percentage of outgoing edge length.

Nodes are connected via *edges*: any number of edges may enter or leave an individual node, with edges represented visually as a solid line with an arrow at the end indicating the direction of the connection (Fig. 2). The way players select edges to traverse when moving from one node to another depends on the node type, detailed in Section 2.4. The time taken to travel along an edge is proportional to its total length. A special kind of edge, known as a *wormhole* instantly transports any agent travelling on it between the nodes to which the edge is connected. Wormhole edges are represented visually as dashed lines (Fig. 2).

By default, edges follow 'city block' pathways maintaining axis aligned edges between nodes. This ensures quantised timing. The edge editor permits changing this behaviour to 'shortest path', permitting edge lengths with irrational ratios.

Lists Nodes and edges may contain *list-based* state-change information. These are linear lists of values, with both absolute and relative values permitted in each list. The owner node or edge maintains pointers to current list values, and each time they are traversed by a player the current list value is used, then the pointer is immediately incremented. Pointers at the end of a list are automatically reset to the beginning.

Lists allow a sequence of notes to be triggered from the same node, leading to more complex harmonies and structures. Lists are provided for pitch, velocity (volume) and duration[1]. In edges they provide a way to change the velocity of agents moving through the edge on a per-edge basis. This allows same-length edges to be traversed in different times.

2.4 Node Traversal

When a player agent arrives at a node, its state information is updated by the data in the node, MIDI notes are normally played and the agent prepares to move on. Any node may have one or more output edges, in the case of more than one output edge the question of which edge to take arises. *Nodal* offers three distinct node types: sequential, parallel and random. The editor symbol and behaviour of each type is show in Table 1.

Table 1. Supported node types

Type	Symbol	Description
sequential		Output nodes are fired one at a time, in the order specified by the output edge sequence editor for the node.
parallel		All output nodes are fired simultaneously. The agent clones itself to produce copies for each output edge.
random		An output edge is selected randomly with weights determined by the output edge sequence editor for the node.

When a sequential node has multiple output edges there must be some way of determining the output order. This is achieved using the output edge sequence editor (Fig. 4). When a node is selected in the network editor, its output edges are automatically assigned different colour codes[2]. The colours of the edges shown in the network editor are duplicated as a sequence of colour chips with an associated

[1] The duration of the note event is determined by the node, the time between events by edge length.

[2] Colours are used for edge differentiation rather than numbers or alphanumeric characters due to the difficulty in dealing with adding or deleting edges while editing,

count for each chip. These chips form the output edge sequence, read from left to right. The chips can be moved, swapped, repeated and the count for each can be changed. This method provides an intuitive way to control complex traversals. For example, the sequence shown in the figure generates the following sequence of output edge traversals each time a player agent enters the selected node:

orange → orange → orange → green → green → blue → purple → purple → purple → purple → blue → blue → green → ...

After completing the last element of the sequence the sequence begins again.

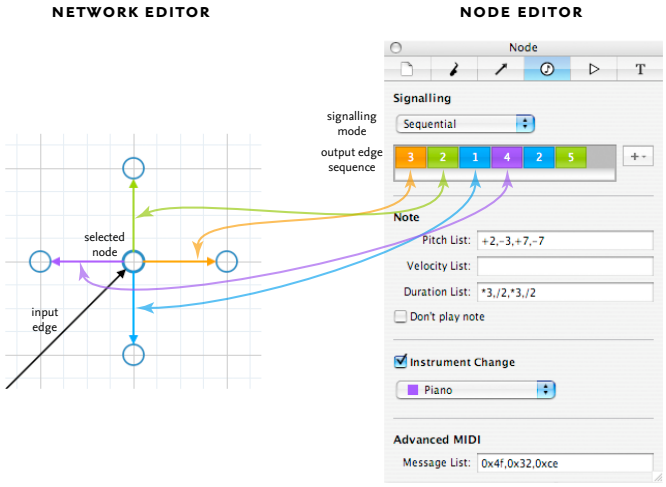

Fig. 4. Sequence editing. Output edges for the node selected in the network editor (left) are colour coded to match the numeric colour chips that specify the sequence of output edges in the node editor (right).

For random nodes the numeric counts for each colour chip become probabilistic weights. For a given node with k output edges, the probability of edge e_i being traversed is:

$$Pr_T(e_i) = \frac{\sum_{i \in e_i} c_i}{\sum_{j=1}^{k} c_j} \tag{2}$$

where c_i is the count value for colour chip i. For parallel nodes the sequences specified by the colour chips are ignored, although the sequences are still shown and maintained, allowing the user to cycle between different node behaviours without loss of information.

which would leave missing elements in sequences. Colour coding also provides a natural differentiation for output edges (Fig. 4).

3 Emergent Structure

In this section we look at the behaviour of certain simple network structures and show how they can lead to complex musical outcomes.

3.1 Linear and Cyclic Structures

The simplest type of network is a linear chain, with each node representing a note in sequence and the distance between nodes the inter-onset time between events (Fig. 5A). This closely mimics conventional linear notation.

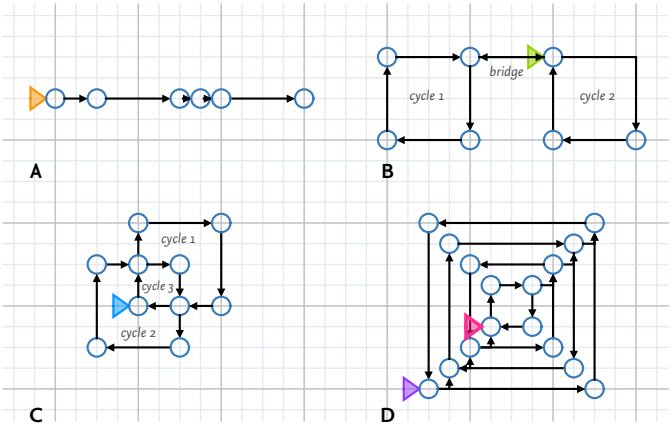

Fig. 5. A linear network (A); two-part cycle with bridge (B); coupled 3-cycle (C); 'recursive' network of connected cycles with player start points at each end (D).

A simple *cycle* consists of two or more sequential nodes with a uni-directional pathway of edges forming a cycle. The example in Fig. 5B shows a 3-node and 4-node cycle connected by a two-way *bridge*. The bridge allows flipping between two different cycles. Adjusting the output edge counts for the two nodes that form the bridge controls the number of repeats of each cycle.

Fig. 5C shows three inter-connected cycles without an explicit bridge (the bridge being the cycles themselves in this case). Combining a structure like this with harmonically related pitch lists generates structures of surprising complexity. Placing additional players on different cycles within the one structure permits temporal interplay between players as each repeats the cycle sequence from different locations on the network. Fig. 5D shows a 'recursive' group of spiralling cycles, each connected into and out of the spiral. The geometric nature of the structure makes the speed of the cycle increase as the spiral moves inward. Placing two players at the outer and inner-most cycles produces a complex interplay of timing — a task that would be difficult or impossible using conventional linear compositional tools, yet simple with *Nodal*.

Bi-directional and Asymetric Cycles In addition to uni-directional cycles shown in the previous examples, bi-directional cycles can also be created. In general, the diversity of behaviour from looped networks that comprise sequential nodes, is related to two factors. Firstly, the extent of interconnection between nodes, with the complexity dependent on the number of edges more than the number of nodes. Secondly, symmetry in arc connections also plays a role. As shown in Fig. 6, a three node graph produces more diversity with an asymmetrical distribution of five edges than occurs with a regular bi-directional graph of six edges.

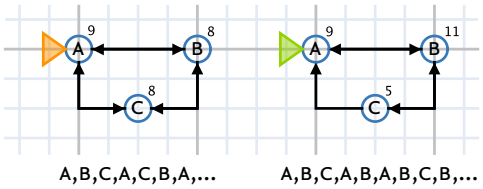

Fig. 6. A bi-directional network (left) produces regular forward-back cycles, whereas removing one edge (right) breaks the symetry and produces more interesting variation. The numbers above each node show the number of times each has been traversed by the player agent.

Cyclic Pitch Phasing with Lists Nodes may contain lists of pitches to change the note played at each traversal, the position selected from the list incremented each time (Section 2.3). Inter-onset time is generated by graph geometry, whereas all other note parameters are drawn from state changes specific to any node. As the list for each parameter can be a different length, it is possible to generate phased combinations of parameters. The de-coupling of inter-onset time from other note parameters results in a wide range of combinatorial possibilities even with the simplest of cycles as shown in Fig. 7. The list of pitches for each node are shown above the node itself.

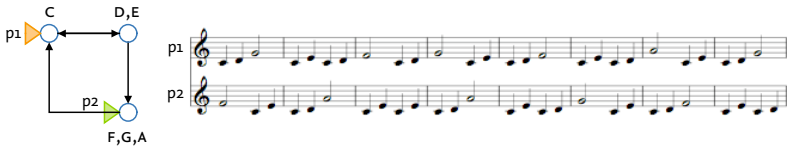

Fig. 7. Output pitches in a loop resulting from a phased combination of pitch parameters. The generated notes form a long string that does not repeat in the short-term.

3.2 Parallel Nodes and Feedback Loops

Parallel nodes can be used for three main purposes: polyphony, synchronisation and feedback. Fig. 8A shows the linear network of Fig. 5A with additional polyphonic components. Parallel nodes trigger additional note events via 'wormhole' edges which allow the player agent to move instantly between nodes, so vertically aligned nodes play simultaneously in the example shown. Fig. 8B shows a simple four-beat cycle, with parallel nodes used for each beat[3]. Each beat in the main loop triggers individual sequences. This highlights the use of parallel nodes for synchronisation where individual nodes are used to synchronise the triggering of sub-sequences. A typical use of this pattern is for rhythmic patterns and drum parts. For example, the main beats in Fig. 8B could be the kick drum, with the sub-patterns high-hats — generating a simple repeating $\frac{4}{4}$ drum pattern.

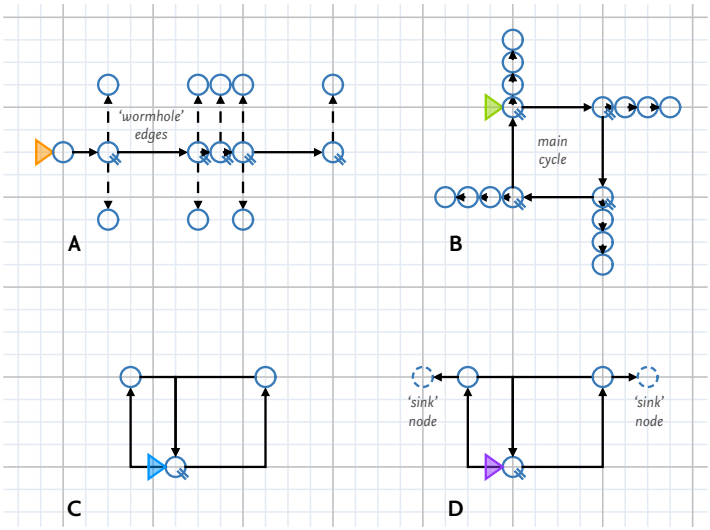

Fig. 8. A linear network with parallel nodes and 'wormhole' edges generating polyphonic sequences (A); a four beat cycle with each main beat triggering event associated with that beat (B); three-node feedback loop (C); three-node feedback loop with 'sink' nodes to prevent infinite feedback (D).

Fig. 8C shows a simple feedback loop. The parallel node at the bottom triggers the two sequential nodes. Due to the differences in distances (the left node is 1.5 beats from the parallel node, the right is 2 beats), the left agent returns to the parallel node first, triggering two new traversals, and so on. The problem with this configuration is that it quickly leads to an exponential expansion in the number of agents traversing the network. The software places an upper limit on

[3] In these examples we assume 1 principle per beat.

the number of agents active at any one time so the network eventually reaches 'saturation' where no new agents can be spawned by the parallel node.

A solution to this exponential positive feedback is to augment the feedback network with 'sink' nodes (Fig. 8D). These nodes provide a terminating pathway for some of the player agents as they feedback through the network. The sink nodes are shown with dashed outlines to indicate they don't trigger any events when an agent arrives (this option is set in the node editor – see Fig. 4). Controlling the ratio of player agents that return into the feedback loop vs. the number that pass to the sink is achieved by setting appropriate values for the output edges in the node editor for each of the two sequential nodes in the feedback loop. For example setting the ratios at 3:2 for the left node and 2:3 for the right generates complex asymmetrical patterns (Fig. 9).

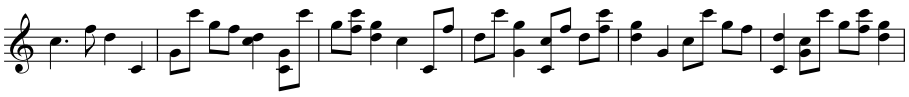

Fig. 9. The first few bars of output from the feedback network with sinks shown in Fig. 8D.

4 Examples

In this section we give some compositional examples of how *Nodal* can be used as a practical compositional tool. *Nebula*, shown in Fig. 10, combines multiple feedback loops with unconventional timing, the network developed without the use of the metrical timing grid. The composition, for single piano, shifts between several single-note codas, punctuated by periods of sudden activity — an explosion of notes as clusters of parallel nodes fire near-simultanously.

Cascades is a more conventional composition, with two parallel nodes driving rhythmic sequences. Multiple players traverse the network each a beat or 1.5 beats behind each other, repeating the sequences which slowly shift phase. The topology of the network results in a cascade of harmonic sequences, the results surprisingly interesting for a network of less than 20 nodes.

5 Conclusions and Future Work

This paper has described a generative composition system that mixes compositional control with the emergent properties of feedback networks. The project addresses the problem of designing dynamic, graphic notation systems suitable for composing and specifying generative algorithmic processes for music composition. Conventional representation systems are ill suited to effectively notating

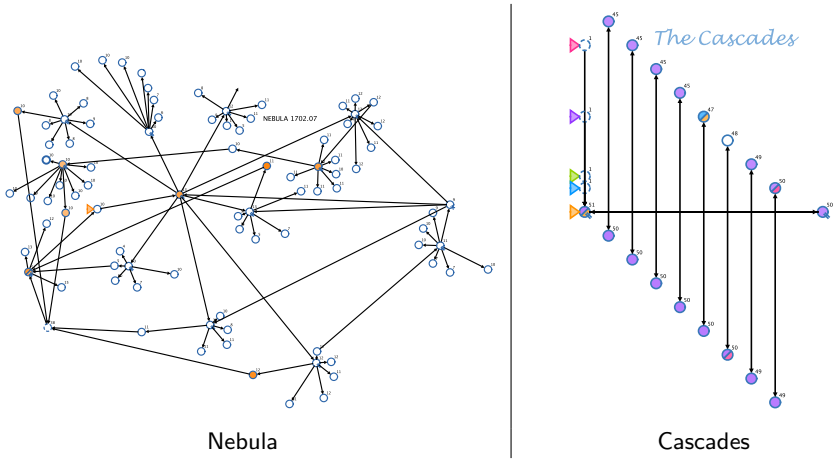

Fig. 10. Two example networks. *Nebula* on the left uses non-linear time ratios and geometric patterns such as logarithmic spirals. *Cascades* (right) creates a complex, shifting, multi-timberal composition with less than 20 nodes.

the musical output of non-linear generative processes. Moreover, they are inadequate for programming specification of such processes. The scheme examined in this paper represents an on-going investigation into how visual programming and notation systems can be designed for generative and Alife musical composition.

There are many directions for further investigation. One important consideration is the limitation of specifying graphs in two-dimensions. This restriction can become problematic when trying to design complex topologies (one of the reasons for implementing the wormhole edge was to circumvent this problem). We have also begun to address the specification of continuous information, by allowing players to interpolate controller information as they travel along edges. This makes possibilities such as pitch bends and portamento possible. Finally, the rigid structure of the metrical grid can make sequences generated by *Nodal* sound mechanical due to their perfect timing. We are working on musically useful ways to circumvent this problem, without losing the benefits that the grid system provides.

Nodal is a compositional tool of enormous possibility, easily capable of generating complex, emergent structures, yet intuitive and simple to understand and control as a composition system. We encourage readers to try *Nodal* for themselves. Download it from `http://www.csse.monash.edu.au/cema/nodal/`.

References

1. McCormack, J., Dorin, A. Art, Emergence and the Computational Sublime. In: Dorin, A. (ed.): Proceedings of Second Iteration, CEMA, Melbourne, Australia (2001) 67–81

2. Whitelaw, M.: Metacreation: art and artificial life. MIT Press, Cambridge, MA (2004)
3. Cariani, P.: Emergence and Artificial Life. In: Langton, C.G. (ed.): Artificial Life II, SFI Studies in the Sciences of Complexity. Volume 10. Addison-Wesley, Redwood City, CA (1991) 775–797
4. Peusner, L.: A graph topological representation of melody scores. Leonardo Music Journal **12** (2002) 33–40
5. Chemillier, M.: Automata and Music. In: Strange, A. (ed.): Proceedings of the 1992 International Computer Music Conference. International Computer Music Association, San Francisco (1992) 370–371
6. Lyon, D.: Using stochastic petri nets for real-time nth-order stochastic composition. Computer Music Journal **19** (1995) 13–22
7. McCormack, J.: Grammar-Based Music Composition. In: Stocker, R. et. al. (eds.): Complex Systems 96: from Local Interactions to Global Phenomena. ISO Press, Amsterdam (1996) 321–336
8. Pope, S.: Music notations and the representation of musical structure and knowledge. Perspectives of New Music **24** (1986) 156–189
9. Guéret, C., Monmarché, N., Slimane, M.: Ants can play music. In: Ant Colony Optimization and Swarm Intelligence. LNCS 3172. Springer, Berlin / Heidelberg (2004) 310–317
10. Wishart, T.: On Sonic Art. New and revised edition edited by Simon Emmerson Contemporary Music Series. Harwood Academic Publishers GmbH, Amsterdam (1996)
11. Lunenfeld, P.: Snap to grid: a user's guide to digital arts, media, and cultures. MIT Press, Cambridge, MA (2000)