Spatio-temporal Patterning in Computer-generated Music: A Nodal Network Approach

Peter McIlwain
Department of Music Studies
pmcilwai@arts.adelaide.edu.au

Anthony Pietsch
Psychology Department
University of Adelaide, South Australia. 5005. Australia.

Abstract In generating computer music for a multi-speaker array, a nodal network has been employed. Here, component nodes are designed as complex behavioural units, modelling behavioural characteristics of neurons. A network of interacting nodes is able to generate evolving patterns of behaviour. Because a node triggers a specific speaker in the array to sound, spatio-temporal patterning occurs. Pitch and dynamic information is generated using flocking algorithms. Individual nodes act as members of a ‘flock’, generating notes or dynamics which are unique to the individual but which follow the generalised path of the group.

1. Introduction
A computer program has been written using the “Max” graphical programming environment. It is designed to generate musical events, displaying spatio-temporal patterning in real time. At the core of the program is a connectionist network consisting of 32 ‘neuron-like’ nodes. These nodes inhabit a virtual space which reflects the actual space of the multi-speaker system.

Activity in the network is the result of ‘firing’ by individual nodes in response to ‘firing’ by other nodes. When any node fires, it immediately causes a note to sound in a corresponding speaker, as well as sending (delayed) signals to all other nodes. For each speaker, one or more nodes are assigned, which output their information on a MIDI channel specific to that speaker.

Spatio-temporal patterning evolves as a result of complex interactions between nodes. A balance between stability and unpredictability is maintained (and controlled) with the use of algorithms that modify the network in response to both the current state and the recent history of activity.

2. Node Design
2.1 Triggering The Node. The triggering mechanism of a node is similar to the action potential of a neuron. A node fires whenever it has received a certain amount of input (from other nodes). An accumulator maintains a running total of input values which have been received. As soon as the total exceeds the ‘trigger threshold’, the node fires and the accumulator is reset to zero.

2.2 Recovery Phase. The recovery phase of the node is also modelled on the neuron. In neural cells, the firing response (or depolarisation) is followed by a period of low sensitivity to inputs. This feature has been incorporated in the design of the nodes, to the extent that they shut down all inputs until a specified period of time has passed.

The recovery phase has a limiting function on the rate of firing of any node. This is sometimes necessary to avoid overloading the computer’s processor.

2.3 Connection Weightings. Connections between nodes act like synaptic connections between neurons. When node A fires, it sends a signal to node B (and every other node), as already outlined. However, the output signal of node A is attenuated as it is transmitted to node B. This is the weight of the connecting signal. If a weight is set to a low value (i.e. the weight WAB is not necessarily the same as WBA) the connecting weightings are continually modified by an algorithm which is responsive to the recent history of the node. It is this algorithm, and the way it is applied, which is larger responsibility for the adaptive and self-regulatory properties of the system. It also provides the most powerful and direct means for generalised control of the system.

2.4 Weighting Modification. Established experimentally-derived models of the mechanisms involved in synaptic modification (Bienenstock, Cooper & Munro, 1982; Bear & Cooper, 1990) have provided a starting point for the modification algorithm. However, in each case, such models have only described synaptic modification as the function of a firing rate. The system represents a role in connection modification which acts in response to individual firing rates.

The solution to this problem emerged from ideas currently being pursued as part of an ongoing
research program into neural and cognitive systems (in which one of the authors is involved). The weighting modification algorithm, described below, captures potential elements which are believed to be part of the organic processes of synaptic modification. (There are certain practical differences, but these are relatively subtle and cannot be addressed here.)

The algorithm governing weighting-modification can be broken down into three components. These are (for a connection from node A to node B) that:

1.) Modification to \( W_{AB} \) occurs when a signal is passed through that connection;
2.) The amount and effect (positive or negative) of the modification depends on the average level of activity of the receiving node (B), over recent time, according to the function described in the next section; and
3.) Weightings are treated differently on any occasion that the input from node A triggers node B to fire (see Section 2.5).

2.5 The Modification Function. The function returns a 'modification increment', \( M \), which is added to the relevant weighting value (\( W \)) each time the signal it is called. \( M \) can be either positive or negative, which means that the sensitivity of node B to signals from node A (for example) may be either increased or decreased. The effect depends on the 'current state' of node B, determined by an average of intervals between recent firings. \( M \) is calculated by a logistic equation, which generates a response of the form illustrated in Figure 1.

![Diagram](image)

Figure 1. A typical weighting-modification curve, showing the roles of the control parameters.

The relevant expression is:

\[
M = a \times \left( e^{b/x} - d \right)
\]

where

- \( x \) is the recent average rate of firings (of the receiving node),
- \( a \) is a measure of the current state of activity of the node at which \( M \) equals zero and hence no modification occurs. It is the point at which the action of the modifier switches from a positive, or damping, mode to a negative, or sensitizing mode. Thus, the firing rates of the nodes fluctuate around the value set by this parameter.
- \( d \) is the demodulation ratio, which determines the proportion of the function which lies above and below the \( x \) axis, as well as the slope of the curve at the break point. In effect, it sets the relative strength of effect of the demodulation and sensitizing actions of the function.

By modifying \( W \), the algorithm has an effect on the number of signals required to trigger the node and, consequently, on the firing rate of that node. Since the function itself depends on the firing rate, there is a causal feedback loop. The nature of the modification function is soft-limiting, continually driving the firing rates of each node towards the specified break point. This maintains the long-term stability of the system, while still allowing short-term variability.

2.6 Triggering and Non-Triggering Inputs. One of the aims of the project is to encourage sustained interactions ('talking') between pairs or groups of nodes. To achieve this, two modules are used, each containing the same modification function (described above) but with different values of \( a \), \( b \) and \( d \). One module deals with the majority of instances, whereas the input value is added to the accumulator but the receiving node does not fire ('non-triggering inputs'). The other calculates a value of \( M \) that is applied to 'triggering inputs', which are those that do initiate a firing. The parameters are set so that a bias is introduced, which favours any connection that delivers a triggering input signal.

Figure 2 illustrates the way that a tension is created between the two modules, as each of the two functions tries to drive the node's rate of activity towards its own break point. The effect is to promote a higher level of activity in nodes that are talking than in nodes that are not involved in any specific interactions at the time. It is this tension which facilitates talking.
3. Network Design

3.1 Network Structure. Despite the fact that each of the 32 nodes is connected to every other node, the network has a certain structure which is defined by different delay times between different pairs of nodes. The intention is that 'virtual distances' (i.e. delay times between the corresponding nodes) reflect actual distances between sound events (within the multi-speaker system). With this design, interactions between nodes which are far apart need to have much longer periods than in the case between neighboring nodes.

It is at the level of network structure that the operating principles deviate from those of the neural model on which the nodes themselves were based. However, in a much looser analogy, the gross structure of the network corresponds to a higher level of cerebral organisation, involving many neurons. A delay time for each connection is stored in a 32 x 32 matrix. However, unlike the connection weightings (which can float freely), they are set before the program runs, thus defining the shape of the structure. While an unlimited number of different structures are possible, exploration of these has only just begun. Each different network structure has its own characteristic behaviours and resonances.

3.2 A Circular Architecture. In its current form, the network consists of 32 nodes arranged in a virtual circle. It can thus generate behaviour for a circular array of 32 speakers. (Groups of neighbouring nodes can share one speaker, however, reducing the number of speakers required to either 16 or 8.) The longest delay time in the system imposes a maximum limit on the time that the system can remain quiet without completely coming to a halt. Therefore, to provide a broader range of long and short delays, virtual distances are exponentially related to actual (circular) distances.

While the circular architecture may not yield the most interesting patterns of behaviour, it was adopted (in the first instance) as its configuration is one of the easiest to understand.

3.3 Initiating Activity. To initiate activity, a starting state must be artificially created. This can be done simply by running a short sequence of externally-driven firings. However, it is helpful to also specify some initial variation in the connection weightings.

Since the network is a purely deterministic system, the starting state determines the ensuing sequence of events. With systems of this type, often referred to as 'complex systems', small changes to the starting state can sometimes result in quite dramatic changes to the subsequent course of events (the 'butterfly effect' - Gleick, 1987).

4. Generating Pitch and Velocity Information

4.1 A Flocking Model. The network can be considered as a group of independent units which interact with one another, giving rise to a collaborative type of behaviour. This is similar to the flocking behaviour displayed by many birds. A single bird in a flock flies in a path unique to that individual whilst at the same time following the common generalised path of the group. To do this each individual must check the position and speed of the other members of the flock and adjust its path accordingly. It must maintain a safe distance from other members of the flock without straying too far from the group.

In the field of computer animation, several approaches to simulating this behaviour have been developed, such as the Boids algorithm outlined by Reynolds (1987). This work gave rise to the idea of using such an approach to generate pitch and dynamic information, using by flocking behaviour in a way that responds to the behaviour of the nodal network.

For this purpose, two modules containing a flocking algorithm are used - one for generating MIDI pitch values and the other for MIDI velocity values. These values are generated in response to firing by the node concerned. In order to 'lead' the flock in a desired direction the user creates (two) guiding centre's which describes the generalised path that the flock will follow.

4.2 Flocking Implementation. For this discussion, the output of a flocking algorithm is labelled F. The algorithm has several stages. In the first, the firing of a node triggers the calculation of a value, m, which is the mean of 5 input values. These inputs are:

McIlwain & Pietsch 314 ICMC Proceedings 1996
(i) the last two values of \( F \) that the algorithm produced,
(ii) the last value of \( m \),
(iii) the value of \( F \) which was generated last time the same node fired, and
(iv) the next value in the guiding contour.

At the second stage the new value of \( m \) is tested to determine whether the (absolute) difference between it and any of the 5 input values is less than the value of a 'safety zone' (another user variable). If it is not within any of the defined restricted ranges, \( m \) is passed out as the new value of \( F \) for the node concerned, and the process is complete.

If \( m \) is detected to be within any of the safety zones, it passes to the third stage. This stage calculates a value that is as close as possible to \( m \), but which satisfies the requirements of stage two. This then becomes the new value for \( F \).

4.3 Flocking Behaviour. In general terms, the behaviour is the result of a tension between the 'inward' and 'outward' forces of the algorithm. The inward force (where the values of \( F \) tend towards the middle) of the flock comes from taking the mean of the 5 inputs. On the other hand, the outward force results from the fact that values of \( F \) cannot be closer to any of the 5 inputs than the regions defined by the safety zones, forcing them away from the middle.

One characteristic of the algorithm's behaviour is that the values of \( F \) do not immediately follow changes in the guiding contour. If there is a sudden leap in the contour (e.g. leaping from a low value to a high one), the values of \( F \) will lag behind, cascading upwards in an effort to catch up. In the opposing case, where there is no change in the guiding contour (i.e. a straight line), the flocking behaviour will tend to settle into a regular pattern. However, this pattern is periodically disrupted, as some nodes fire less often than others.

5. Discussion

A detailed characterisation of the system's behaviour falls beyond the scope of this paper. Here, a few brief comments must suffice.

The structural configuration and the parameter settings consistently determine certain general qualities of the system's behaviour. To date, this behaviour has included spatio-temporal patterning and periods of sustained talking between nodes. On numerous occasions, patterns have not only evolved but moved and transposed through the virtual space. However, there are indications that even more interesting and organised behaviour will be possible, with more complex node configurations and with additional fine tuning of the operating parameters.

With the type of behaviour exhibited by a system such as this, it is inappropriate to exert precise, moment-to-moment control of specific events. Instead, a strategy of guided or 'loose control' has been adopted. In this way, the user can concentrate on global control, while leaving the system to deal with the detail. Loose control can be useful in many practical applications (e.g. McIlwain, 1995).

Although it can be classified as connectionist, the present approach constitutes a significant departure from the typical application of neural networks (e.g. Todd & Loy, 1991). Here, the output is the behaviour exhibited by the network, rather than a solution to a problem of computation. Classical networks have nodes that act as 'computational units' with connections that are modified by an externally-applied goal-directed training rule. In contrast, our approach has been to design the nodes as complex 'behavioural units', by modelling neural characteristics such as the action potential and a realistic connection-modification rule.

Acknowledgements

We would like to thank Mitchell Whetnall for his ideas, interest and ongoing support for our work.

References


