A System for Growing Graphs with Applications to Reservoir Computing

Riversdale Waldegrave

Workshop on Unconventional Computing, Erice, 20th-26th October 2022

Abstract

A system to grow **directed graphs**, which combines elements of RBNs, Cellular Automata and L-Systems with **graph transformation** techniques. Salient features are: variable degree nodes with fixed arity functions; real-valued matrix-based encoding of state transition rules; potential for natural halting of growth. It is hoped that this can be used in an **evo-devo** system to create effective networks for use in **reservoir computing**.

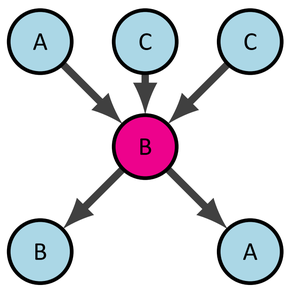
1. Background

- **Cellular Automata:** Each cell updates its state based on the states of cells in its neighbourhood. The cells are arranged in a grid but can be thought of as nodes in a regular graph.
- **Random Boolean Networks:** Each node updates its state using a Boolean function of the states of its neighbours. The network may be irregular, but all nodes have the same degree so that all update functions have the same arity.
- **L-Systems:** Each symbol in a string is replaced with one or more other symbols, allowing the string to grow. Context-sensitive L-systems use replacement rules which take into account the neighbouring symbols.

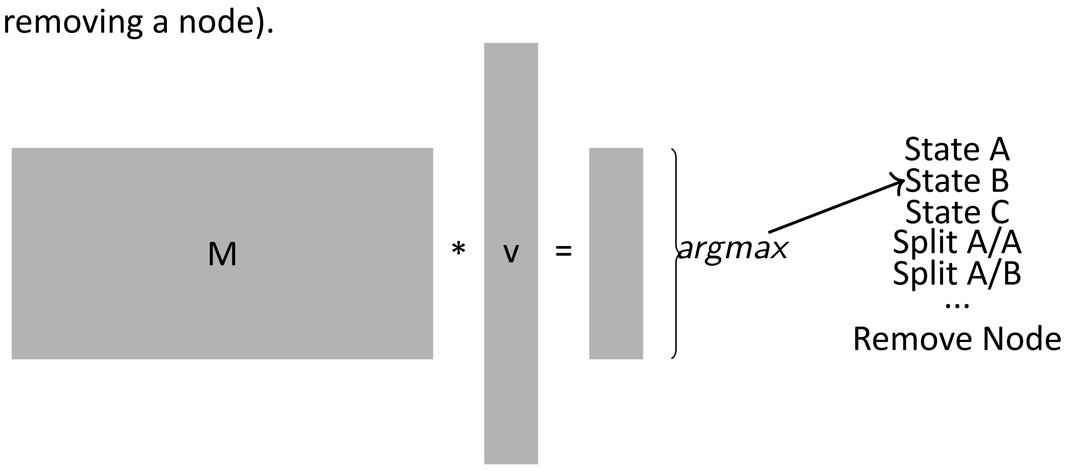
The present system is based on irregular graphs, like RBNs, but transition functions are based on *counts* of nodes in each state, like CAs, rather than ordered lists of node states (RBNs). This means that the transition function can have a fixed arity even though the nodes have variable degree.
The transition rule can change a node's state, but can also add and remove nodes, as in an L-System.

2. Description

- The system starts with a **seed graph** - this is equivalent to the initial configuration of a CA or the "axiom" of an L-System.
- Each node can be in one of a finite set of states Σ
- The transition function at each node takes as input the counts of neighbouring nodes in each state. Predecessor and successor nodes are treated separately. For a three-state system $\Sigma = \{A, B, C\}$, the input vector at each node takes the form: $[p_A, p_B, p_C, s_A, s_B, s_C]$, where p_X is the count of predecessor nodes in state X and s_X is the count of successor nodes in state X.



The input to the transition function at the highlighted node would be: $[1, 0, 2, 1, 1, 0]$. This vector will always have the same length whatever the degree of the node.
The input vector **v** is multiplied by the *rule matrix* **M** to choose the transition to apply. This can be a simple change of state or a structural alteration (adding or removing a node).



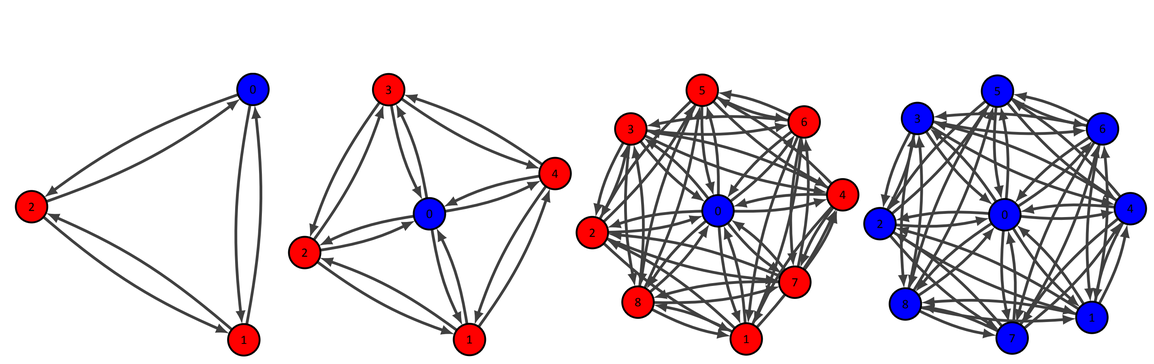
A different rule matrix **M** is used for nodes in each state (ie. a three-state system would have three different rule matrices).

3. Evolving Transition Functions

- The transition function is the genome and is defined by a **real-valued matrix**, making it easy to evolve.
- **Point mutations:** Change a single number in the matrix.
- **Crossover:** Average two matrices, or use half the values from each parent.
- **Particle Swarm Optimisation** is also an option. Move each individual towards the *best* individual's matrix values.

4. Stopping Growth

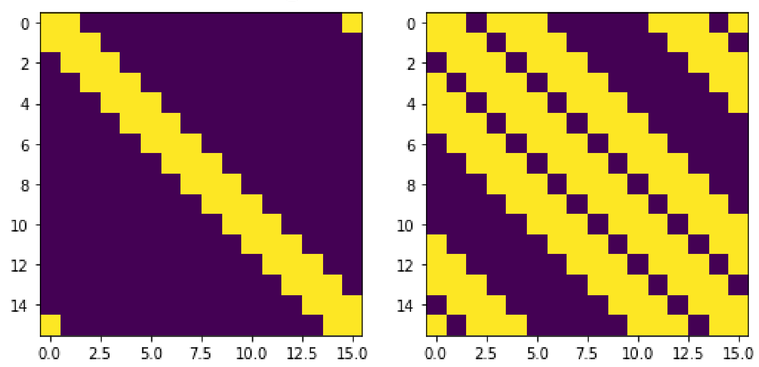
Many transition rules result in continual growth (often exponential!), but some rules result in growth which "naturally" stops - the system enters an attractor in the state space.
In the two-state system below, a seed graph (three fully-connected nodes) reaches an attractor after three timesteps.



5. Reservoir Patterning

In the Reservoir Computing literature, a range of topologies are used, including lattices, rings and tori. These produce varying degrees of reservoir quality (as measured by eg. memory capacity). The present system allows more complex topological patterning so that other structures can be explored.

Adjacency matrices for two standard topologies used in the literature, the ring and the torus.



Two more complex topologies generated by the present system: