Creating Network Motifs with Developmental Graph Cellular Automata

Riversdale Waldegrave[1], Susan Stepney[1] and Martin A. Trefzer[2]

[1]Department of Computer Science, University of York, UK
[2]School of Physics, Engineering and Technology, University of York, UK
{rraw500, susan.stepney, martin.trefzer}@york.ac.uk

Abstract

Analysing network motifs is a common way of characterising biological networks. Motifs are small subgraphs that are more abundant in the observed network than would be expected in random graphs. They may play an important role in network function, and as such may be selected by evolution. In some cases, such as neural networks, they are instantiated via a developmental process. The processes used to structure Artificial Neural Networks, whether training or evolution, do not usually result in motifs or modularity more generally. We introduce a new version of Developmental Graph Cellular Automata (DGCA) which can be used in an evolutionary and developmental (evo-devo) process to produce networks with specific motif profiles. We evolve developmental rules (the "genome") so that networks are produced with similar motif profiles to specific biological networks. Networks produced in this way may have useful computational and/or dynamical properties when deployed as Recurrent Neural Networks (RNNs) or in Reservoir Computing (RC).

Introduction

Networks are found in living organisms in many forms: neural networks, metabolic networks, gene regulatory networks (GRNs), signal transduction networks, and more. Whilst Artificial Neural Networks (ANNs) are inspired by the first of these, studying the "biochemical connectionism" of the others may provide additional insight into how network structure can affect function (Lones et al., 2013).

Traditional network science metrics such as degree distribution, average path length and so on can be a useful starting point for describing these networks, but can get us only so far in actually understanding how they work. The field of Systems Biology has made significant progress in analysing the microstructure of biological networks and how it relates to network function using the concept of network motifs (Alon, 2007). In neural networks we also see larger-scale repeated structures, for example in cortical micro-columns, which may form functional modules (Bennett, 2020). Repeated structure, from small scale motifs to larger modules, may confer benefits on the network.

Neuroscientists such as Hiesinger (2021) have highlighted the large gap between biological neural networks and ANNs in this regard. The former are rich in structure with a developmental process "unfolding" genetic information into the final network. In conventional feedforward ANN architectures, the structure is predetermined and fixed, and the weights are trained. Training procedures such as backpropagation usually do not result in any repeating network structures. When the network structure is the result of an evolutionary algorithm, as in Topology and Weight Evolving ANNs (TWEANNs) such as NEAT (Stanley and Miikkulainen, 2002), modularity also does not typically arise. Indeed, even if the network is seeded with modular structure it normally disappears over the course of evolution, as there are likely to be mutations which increase fitness and break modularity (Kashtan and Alon, 2005).

Although breaking modularity may increase fitness for a particular task, there is ample evidence to suggest that modular structure may confer other benefits, such as robustness, generalisation, interpretability and evolvability. For example, in the field of Reservoir Computing (RC), Wringe et al. (2023) find that Restricted ESNs, which have a degree of enforced modularity by having sub-reservoirs with sparse interconnections, can improve performance. Kashtan and Alon (2005) find that networks with modular structure are more able to evolve, as it can take very few mutations to reconfigure the interactions between modules, allowing higher-level functions to be composed out of subroutines. This can also make it easier to interpret the network structure. Recurrent Neural Networks (RNNs), can be viewed as discrete dynamical systems, particularly when used in Reservoir Computing (RC). Network motifs can affect the dynamics of these systems: Prill et al. (2005) find that different 3-node subgraphs (triads) confer differing degrees of stability on the network, with some encouraging static or oscillatory behaviour whilst others are more prone to chaotic behaviour. In RC such properties may impact higher level functionality of the network such as kernel rank, generalisation rank and memory capacity, metrics often used to characterise RC substrates (Dale et al., 2019). Alon (2007) discusses how particular biochemical network motifs can give rise to lasting or fading memory of input signals, proper-

ties which are highly relevant to RC, which relies on fading memory to ensure the Echo State Property (Jaeger, 2001).

Here we introduce a way to create networks with far-from-random motif distributions using an evolutionary and developmental process based on Developmental Graph Cellular Automata (DGCA) (Waldegrave et al., 2023a), a bio-inspired model of development where each network node uses only local information to determine its developmental steps with reference to fixed rules (the common "genome").

The paper is organised as follows. We review existing methods of creating networks with modular structure, describe how evolutionary and developmental processes may influence the occurrence of motifs in biological networks, and describe a commonly used method of motif detection. We then introduce DGCA-M, used to model growing networks with an evolvable growth rule. We then present the results of two sets of experiments that demonstrate the ability of DGCA-M to grow networks with a range of different motif profiles: the first uses random search over growth rules; the second uses-goal directed evolution towards a particular distribution of motifs based on real biological networks.

Background

Generating Networks with Structure

The most basic way of creating a random network is the Erdős–Rényi model, which requires only two parameters: N, the number of nodes and p the probability of a connection between any pair of nodes (Erdős and Rényi, 1959). That model does not tend to create networks with repeated structure, and instead is used as a baseline against which motif occurrence is measured.

An elaboration of the Erdős–Rényi model is the Second-Order Network (SONET) model (Zhao et al., 2011). It defines four two-edge subgraphs (reciprocal, convergent, divergent and chain) whose probabilities can deviate from the independent edge probability p. For instance, whereas the independent probability of reciprocal edges between two nodes is p^2, this can be modified by the parameter α_{recip} to $p^2(1 + \alpha_{recip})$. There are four such α parameters that modify the probabilities of the four two-edge subgraphs. This approach is likely to be effective only for very small motifs, as the higher order correlations between edge probabilities quickly become very complex with larger motifs.

An alternative approach is to have a predefined library of larger motifs or modules that can be composed to produce the final network. This is the approach taken by Walter et al. (2023), who use a genetic algorithm to find effective combinations of predefined modules. This works well in the context of electronic circuits where it makes sense to have a predefined library of blocks. However, it is difficult for new motifs or modules to emerge.

One approach that encourages the spontaneous emergence of network structure during the evolutionary process is described by Kashtan and Alon (2005). During evolution, they regularly switch between two target functions which are easily decomposed into shared subfunctions. For example, they use the two Boolean logic functions: (X XOR Y) AND (Z XOR W) and (X XOR Y) OR (Z XOR W). By switching between these two targets, they encourage the networks to develop two submodules implementing the XOR function, which can then be combined in different ways. This approach relies on having target functions that can be easily decomposed, and it is not clear how it would scale to more complex tasks.

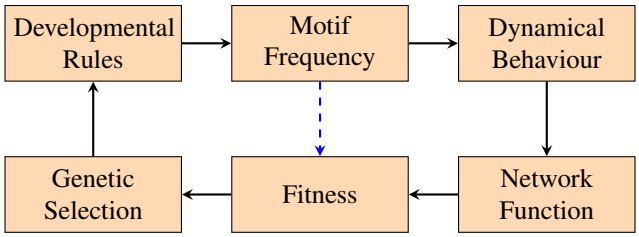

Figure 1: The chain of causation implied by combining the two hypotheses (motifs are evolved / due to developmental constraints) into an evo-devo process. The dashed arrow shows the simplified loop used in experiments here: we search directly for motif occurrence rather than assessing the functional properties of the network.

Evo-Devo Approach

It is hypothesised that the over-representation of certain motifs in biological networks is due to evolutionary selection: abundance or lack of particular subgraphs affects the dynamical behaviour of the network, which in turn affects the function of the network, ultimately influencing the fitness of the organism (Prill et al., 2005). Eom et al. (2006) show that the motif profiles of metabolic networks are more similar to each other in organisms from the same taxonomic group, supporting the claim that evolution is responsible for the distribution of subgraphs. Shellman et al. (2014) claim that analysis of network motifs of metabolic networks can provide information about the evolutionary origin of organelles.

An alternative hypothesis (also in Prill et al. (2005)) is that there are certain constraints on the development of the network that lead to over- or under-representation of particular motifs. Leier et al. (2007) support this view, arguing that the particular far-from-random motif distributions observed in GRNs are a consequence of the processes of network generation (gene duplication followed by functional divergence) rather than evolutionary selection. From an evo-devo standpoint these two views are not incompatible. It may be that the rules or constraints on network development are the direct cause of motif abundance, but that these rules are themselves genetically encoded and selected by evolution. The full chain of causation for this combined hypothesis is shown in Figure 1.

In this paper, we used a simplified version of this loop, basing our fitness evaluation directly on the motif distribu-

tion rather than actually assessing the behaviour of the network. The aim is to assess the ability of our system to produce networks with a wide range of different motif profiles, rather than to optimise a network for a particular task.

We build on Developmental Graph Cellular Automata (DGCA) Waldegrave et al. (2023a) to model the development of networks. The DCGA update rules (encoded as weights) constitute the "genome", which controls development of the network from a small seed graph (which can be a single node) into a large network. Development can be stopped after an arbitrary number of time steps or at a fixed network size, or it can automatically come to a halt as the system enters an attractor. This may be a point attractor, where the network remains in a static configuration, or a cyclic attractor, where the network can cycle through two or more configurations (Waldegrave et al., 2023b). We extend DCGA to include the capability to create networks with repeating structure.

Motif Detection

A k-node subgraph is termed a "motif" if statistically over-represented in a particular network. Subgraphs that occur less than expected are "anti-motifs". Following Milo et al. (2004), Prill et al. (2005), Shellman et al. (2014) and others, we consider three-node subgraphs. There are thirteen possible node-induced connected three-node subgraphs, or triads as shown in Figure 2 (throughout the analysis self-loops are not considered[1]).

We follow Milo et al. (2002) to determine whether a triad is under- or over-represented in a network. Since the degree distribution of a graph might have a large impact on the occurrence of motifs, the count of the triads in the observed network is compared with the counts in an ensemble of randomised graphs with the same degree distribution. The ensemble is generated using a shuffling procedure: for each edge in the network, its source or target is swapped with the source or target of another randomly selected edge.

The Z-score of triad i is calculated as:

$$z_i = \frac{c_i^{real} - \langle c_i^{rand} \rangle}{std(c_i^{rand})} \tag{1}$$

where c_i^{real} is the count of triad i in the real network, and $\langle c_i^{rand} \rangle$ and $std(c_i^{rand})$ are the mean and standard deviation of the counts of triad i in the ensemble.

Treating the collection of Z-scores (13 of them when $k = 3$) as a vector $\mathbf{z}$, the length (L2 norm) of this vector can summarise in a single number how atypical the motif count distribution of the observed network is compared to the randomised networks.

$$\|\mathbf{z}\|_2 = \sqrt{\sum_j z_j^2} \tag{2}$$

[1]There are 86 possible 3-node motifs with self-loops, only 13 without self-loops. Excluding self-loops also follows the literature: although this is not explicitly stated in Milo et al. (2002), it is evident from the reported size of networks used.

Following Milo et al. (2004) we normalise the vector of Z-scores to get the "significance profile" vector $\mathbf{s}$:

$$\mathbf{s} = \mathbf{z}/\|\mathbf{z}\|_2 \tag{3}$$

These normalised Z-scores range from -1 to 1. Triads with positive normalised Z-scores (or greater than some threshold) can be considered motifs of the network, whilst those with negative scores can be considered anti-motifs. The vector $\mathbf{s}$ is known in the literature as the Triad Significance Profile (TSP) and can be used to characterise a network. Figure 3 shows the TSPs of four biological networks.

A key step in the procedure described above is counting the number of k-node subgraphs (in the observed network and in the ensemble of randomised ones). This has received considerable attention in the literature, as it is computationally intensive. We use the RAND-ESU algorithm (Wernicke, 2006), as implemented in the graph-tool Python library (Peixoto, 2014). This uses an unbiased random sample of k-node subgraphs from the input graph. We sample 50% of the 3-node subgraphs, with sampling rates of $[1, 1, 0.5]$ at each level of the ESU tree (see Wernicke (2006, Fig.4) for an explanation). These values were chosen after a parameter sweep to determine a good trade-off between significance profile stability and speed.

Developmental Graph Cellular Automata

Here we describe our extended version of the Waldegrave et al. (2023a) DGCA system. Our DGCA-M can create more complex graph structures, including graphs with a wide range of motif significance profiles.

Unlike a traditional CA, which has cells arranged on a grid, in a Graph CA cells are nodes in a graph, with an arbitrary topology, usually fixed in advance. The nodes in a Graph CA may have different numbers of neighbours, making it difficult to use a lookup-table state transition rule. One approach is to use *outer totalistic* rules: simply count the neighbouring nodes with different states, so it does not matter how many neighbours there are (Hill et al., 2005; Marr and Huett, 2009). Another option is to use a small neural network for the transition rule (Grattarola et al., 2021).

DGCA takes the latter approach, but in addition to updating the node state, the update process for each node can choose an "action", including removing the node from the graph or "dividing" to create a new node. In this way, the structure of the graph can change over time. All of these actions are decided at the level of the individual nodes, using only local neighbourhood information: there is no centralised control of the development of the graph.

In this respect, DCGA has parallels with biological morphogenesis. In particular, it takes inspiration from juxtacrine signalling, where signals are passed between cells in direct contact, as opposed to longer range signalling by diffusion or circulation of signalling molecules (paracrine/endocrine).

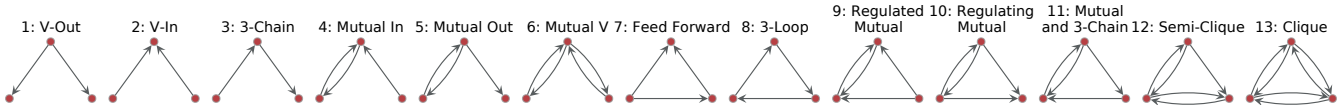

Figure 2: The 13 possible three node subgraphs (triads), which are used in experiments in this paper. Triad numbering and names follow Milo et al. (2004) and Shellman et al. (2014). Subsequent figures refer to the triads by the numbers here.

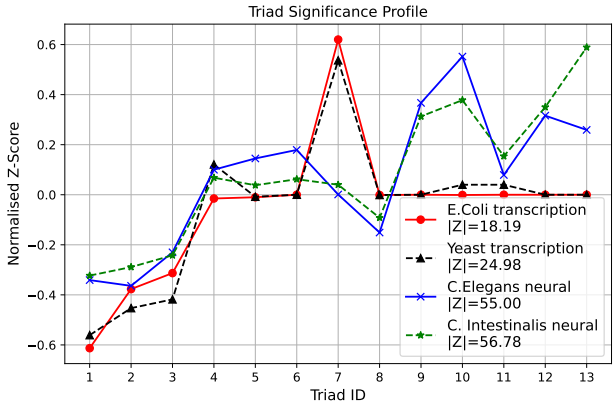

Figure 3: Triad significance profiles of some biological networks. Although the x-axis data is categorical we link the points with a line for visual clarity, in effect treating each TSP as a vector in 13-dimensional space and using the parallel coordinate system of Inselberg (1985). As noted in Milo et al. (2004), TSPs can be used to group networks into families: the two transcription networks have similar TSPs to each other, as do the two neural networks. Note that $|\mathbf{z}|$ is higher for the neural networks, indicating that their triad distribution is more atypical (compared to random networks). The data for the *E.Coli* and Yeast transcription networks were compiled by Milo et al. (2002) The data for the *C.Elegans* and *C.Intestinalis* neural networks were compiled by White et al. (1997) and Ryan et al. (2016) respectively. In all cases, data were downloaded from https://networks.skewed.de/.

The Notch signalling pathway is an example of juxtacrine signalling that is thought to be particularly important for neurogenesis (Luo, 2016).

DCGA with motifs: DGCA-M

The original DGCA system has one single layer perceptron (SLP) to perform the node state update and the node action choice. In DGCA-M we divide the SLP into two parts. This is similar to the Neural Development Programs introduced by Najarro et al. (2023), in which one multi-layer perceptron (MLP) is used to update the node state and another to add extra nodes. A more significant innovation in DGCA-M is that when a node divides, there are various options for how it connects to neighbouring nodes. The chosen configuration is also specified by the output of the action-choice SLP, as explained below. A schematic of one DGCA update cycle is shown in Figure 4 and consists of the following steps:

Neighbourhood Information Aggregation: The input to each of the SLPs is a vector of neighbourhood state information. This includes the node's own state and the counts of incoming and outgoing neighbours in each state. Consider a system with s states and a graph that currently has n nodes. We record node states as one-hot encoded vectors stacked into a matrix $\mathbf{S} \in \{0,1\}^{n\times s}$. We gather all the neighbourhood information by a series of matrix multiplications with the graph adjacency matrix $\mathbf{A} \in \{0,1\}^{n\times n}$.

$$\mathbf{C_{out}} = \mathbf{A}\cdot\mathbf{S}; \quad \mathbf{C_{in}} = \mathbf{A}^T\cdot\mathbf{S} \tag{4}$$

The count of outgoing and incoming neighbours in each state is given by $\mathbf{C_{out}}$ and $\mathbf{C_{in}}$ respectively (both $\in \mathbb{N}^{n\times s}$). These are horizontally concatenated with the original S (to indicate each node's own state) to give the neighbourhood information matrix $\mathbf{G}$ (component matrix sizes are shown as subscripts):

$$\mathbf{G}_{(n\times 3s)} = [\mathbf{S}_{(n\times s)}|\mathbf{C_{in}}_{(n\times s)}|\mathbf{C_{out}}_{(n\times s)}] \tag{5}$$

Each row of $\mathbf{G}$ gives the neighbourhood information vector of one node in the graph.

Graph Structure Update Function: This neighbourhood information $\mathbf{G}$ is passed through an SLP with an input layer of size $3s+1$ (the $+1$ is for a bias) and an output layer of size 15. The output vector subdivided into four sections. The *argmax* of the first three entries determine the *action* that the node should take: it can choose between removing itself from the graph, staying as it is, or dividing. If the chosen action is removal or stasis, we do not need to look at the remaining entries. However, if the node chooses to divide, the remaining entries specify which edges the newly created node should have.

To preserve localism, the new node is not allowed to connect to remote nodes. The set of nodes that it "knows about" includes the parent node itself, and any nodes that are directly connected to the parent node (with either edge direction). However, if these were the only nodes that an offspring could connect to, it would not be possible for subgraph structures to be recreated. Any offspring nodes created at time t could only connect to nodes that were already in existence at time $t-1$ (their "parents' generation").

To create the potential for whole subgraphs to be duplicated, we add to the set of valid edges connections with "cousin" nodes: other offspring nodes that are created at

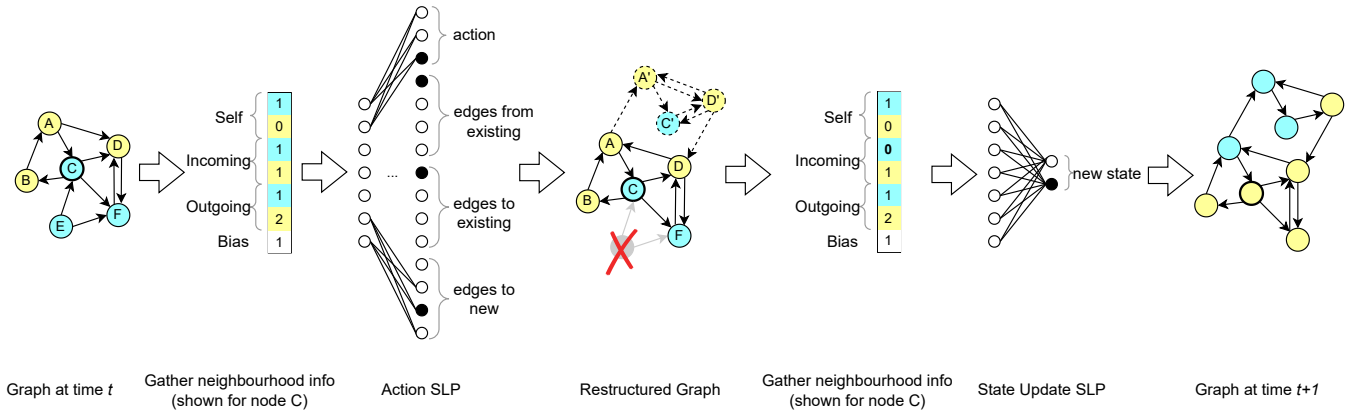

Figure 4: A schematic of the DGCA update procedure in a two-state system. All nodes update in parallel; here the neighbourhood information vector is shown just for node C. The Action SLP determines that nodes A, C and D divide and node E is removed. The dividing nodes choose to replicate connections amongst themselves (except that one edge is reversed). A chooses to create an edge from its parent, and D to its parent. See Table 1 for more detail on the output of the Action SLP. Once the structural update is complete, neighbourhood information is again gathered (node C now has zero incoming blue neighbours since D was removed), and passed through the State Update SLP.

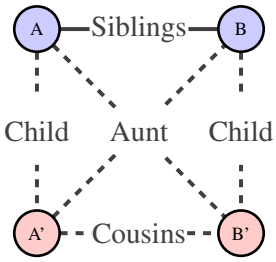

Figure 5: Node A and B are connected at time t (so are "siblings"). They each divide at time $t+1$, creating child nodes A′ and B′. These nodes can have connections to their parents, the neighbours of their parents ("aunts") and any children of their aunt also created at time $t+1$ ("cousins").

the same timestep, where there is a connection between the parents (who are therefore "siblings"). This is shown in Figure 5. Nodes have no way of predicting whether their sibling nodes will divide, so when cousin edges are specified, they are only *potential* connections. For example, in Figure 5, offspring node A′ can specify that it wants a connection to offspring node B′, but this is instantiated only *if* B′ exists. We judge that the creation of an edge between A′ and B′ does *not* break the localism rule, that it does not constitute a connection to a "remote" node. It is as if the two parent nodes A and B have been cloned along with their edge A–B.

Figure 6 shows all the valid edges for a newly created node. It can have an edge from or to its parent node (red edges in Figure 6a and 6b respectively). It can have the same incoming or outgoing edges as its parent (green edges in Figure 6a and 6b). It could also have the reverse of its parent's incoming and outgoing edges (blue edges in Figure 6a and 6b). Furthermore, it could have edges to other newly created nodes, as shown in Figure 6c: to itself (red), to the children of its parent's outgoing nodes (ie. cousins, shown in green), or to the children of its parent's incoming nodes (blue). Allowing new nodes to have some subset of these edges preserves localism whilst allowing complex network structure (including repeated subgraphs) to emerge.

In order to prevent the graph becoming too dense (since biological networks tend to be very sparse), we allow each new node to have only *one* set of edges (red, green or blue) from each of the three categories: edges from existing nodes, edges to existing nodes, edges to newly created nodes. Whichever set of edges is chosen within each category, the new node gets *all* of those edges. A fourth possibility is to have no edges in a category. This process is illustrated in the last 12 rows of Table 1: there are four choices within each of the three categories.

The last 12 entries in the Action SLP output vector are correspondingly divided into three groups of four. The *argmax* within each group specifies which set of edges from that group the new node should have. An example output for three of the nodes in Figure 4 is shown in Table 1.

In the example shown in Figure 4 the subgraph ACD has been replicated in full, with the addition of two extra edges (D′ → A′, D′ → C′). This highlights how the system is able to replicate subgraphs of arbitrary size, while also introducing variation to the copies. All the "decisions" are made at the level of the individual nodes using only local information: there are no predefined modules or groups of nodes which take action together. It should therefore be capable of creating the repeating structures (with variation) that we see in biological networks, and do so in a somewhat "bioplausible" way.

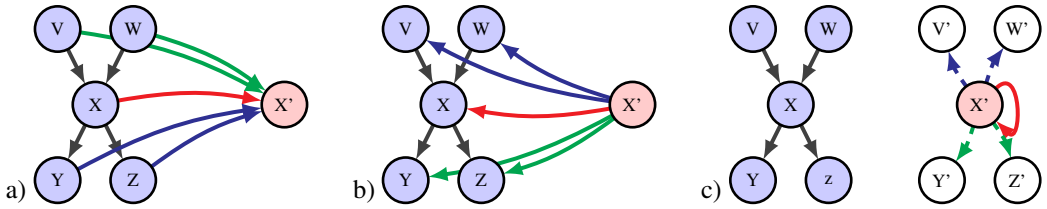

Figure 6: All valid edges choices for newly created node `X'`, offspring of node `X`. (a) The three sets of possible incoming edges from existing nodes. (b) The three sets of possible outgoing edges to existing nodes. (c) The three sets of possible outgoing edges to new nodes created in the same timestep (if they exist: edges to "cousin" nodes are dashed to show that they are only "potential edges"). Red edges are direct connections to/from the parent or the node itself; green edges are equivalent to what the parent node had; blue edges are the reverse of what the parent had.

		NodeC	NodeD	NodeE
Action	Remove	0.25	0.47	**0.87**
	Keep	0.11	0.23	0.64
	Divide	**0.38**	**0.72**	0.05
From Existing	None	**0.87**	**0.92**	*0.94*
	Parent ($\mathbf{k_{fi}}$)	0.45	0.26	*0.73*
	Parent Incomers ($\mathbf{k_{fa}}$)	0.22	0.32	*.25*
	Parent Outgoers ($\mathbf{k_{ft}}$)	0.70	0.64	*-2.57*
To Existing	None	**0.79**	0.44	*0.53*
	Parent ($\mathbf{k_{bi}}$)	0.02	**0.66**	*-0.73*
	Parent Outgoers ($\mathbf{k_{ba}}$)	0.67	0.23	*0.81*
	Parent Incomers ($\mathbf{k_{bt}}$)	0.27	0.02	*1.22*
To New	None	0.34	0.31	*-3.73*
	Self ($\mathbf{k_{ni}}$)	0.54	0.37	*0.05*
	Equivalent ($\mathbf{k_{na}}$)	**0.86**	0.40	*0.38*
	Reverse Equivalent ($\mathbf{k_{nt}}$)	0.48	**0.78**	*0.49*

Table 1: Illustrative Action SLP output vectors for the nodes marked `C`, `D` and `E` in Figure 4. Nodes `C` and `D` are to divide, whereas node `E` is to be removed (*argmax* of first three entries, highlighted in yellow). The rest of the output vector for node `E` can be ignored. The newly created node `C'` gets no edges from or to previously existing nodes, but does get edges to any newly created nodes that are the offspring of its parents outgoers ("cousins"). Node `D'` gets no edges from previously existing nodes but does get an edge to its parent. It also gets "cousin" edges, but these are reversed (ie. `D'` $\rightarrow$ `A'` rather than the `A` $\rightarrow$ `D` of the parents' generation.)

Node State Update Function: The final step of the update (Figure 4) is to update the node states. The node neighbourhood information vectors are generated in the same way as before and passed through the State Update SLP, which has $3s + 1$ inputs and s outputs. The *argmax* of the output gives the node's new state. Like the action step, this state update is done in parallel for all nodes. This step is the same as a state update in a Neural Graph CA (Grattarola et al., 2021).

Implementation. DGCA-M can be run entirely using matrix multiplication, along with Kronecker tensor products to perform the graph restructuring step. We represent the graph using the adjacency matrix $\mathbf{A}_{(n\times n)}$ and the state matrix $\mathbf{S}_{(n\times s)}$: these two matrices are updated every timestep. This makes the system fast to run, even for large graphs. The graph restructuring function can be defined using Kronecker tensor products. We start by defining four 2×2 matrices which indicate each of the four quadrants of a matrix, with a single entry of 1.

$$\mathbf{Q_m} = \begin{bmatrix}1 & 0\\ 0 & 0\end{bmatrix}, \mathbf{Q_f} = \begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix}, \mathbf{Q_b} = \begin{bmatrix}0 & 0\\ 1 & 0\end{bmatrix}, \mathbf{Q_n} = \begin{bmatrix}0 & 0\\ 0 & 1\end{bmatrix} \quad (6)$$

$\mathbf{Q_m}$ indicates the upper left quadrant, which contains the original adjacency matrix; $\mathbf{Q_n}$ indicates the lower right quadrant, which contains the new nodes; $\mathbf{Q_f}$ indicates the upper right quadrant, which contains the links forward from the original nodes to the new nodes; $\mathbf{Q_b}$ indicates the lower left quadrant, which contains the links backward from the new nodes to the original nodes. We then define

$$\begin{aligned}\mathbf{A}' = &\ \mathbf{Q_m} \otimes \mathbf{A}\\ &+ \mathbf{Q_f} \otimes (\mathbf{I}_n \cdot \mathrm{diag}(\mathbf{k_{fi}}) + \mathbf{A} \cdot \mathrm{diag}(\mathbf{k_{fa}}) + \mathbf{A}^\mathsf{T} \cdot \mathrm{diag}(\mathbf{k_{ft}}))\\ &+ \mathbf{Q_b} \otimes (\mathrm{diag}(\mathbf{k_{bi}}) \cdot \mathbf{I}_n + \mathrm{diag}(\mathbf{k_{ba}}) \cdot \mathbf{A} + \mathrm{diag}(\mathbf{k_{bt}}) \cdot \mathbf{A}^\mathsf{T})\\ &+ \mathbf{Q_n} \otimes (\mathrm{diag}(\mathbf{k_{ni}}) \cdot \mathbf{I}_n + \mathrm{diag}(\mathbf{k_{na}}) \cdot \mathbf{A} + \mathrm{diag}(\mathbf{k_{nt}}) \cdot \mathbf{A}^\mathsf{T})\end{aligned} \quad (7)$$

where $\mathbf{Q_m} \otimes \mathbf{A}$ indicates that the original adjacency matrix, $\mathbf{A}$ (size $n \times n$) is placed into the upper left quadrant of a $2n \times 2n$ matrix, with the other three quadrants filled with zeros. The nine different $\mathbf{k}$ vectors indicate how new nodes should be connected. They are effectively horizontal slices through the output matrix of the action update SLP which is partially illustrated in Table 1 (after it has been transformed to 1s and 0s by taking *argmax* of the column sections). For example, $\mathbf{k_{ni}}$ indicates whether the new nodes should have a self connection (the line "To New: Self" in Table 1). The meanings of the other $\mathbf{k}$ vectors are also marked in Table 1, with the colours corresponding to the sets of edges in Figure 6. These vectors are diagonalised and multiplied by the relevant matrices: the identity matrix $\mathbf{I}$ to create connections from/to parent nodes; the original adjacency matrix $\mathbf{A}$ to create the same connections that the parent node had; the transposed adjacency matrix $\mathbf{A}^\mathsf{T}$ to create the reverse of the parent connections. This is done three times to create the three categories of edges (from existing, to existing, to new),

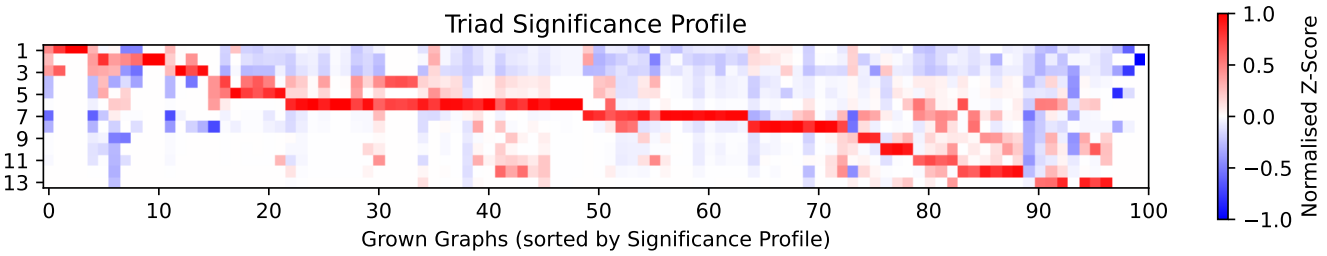

Figure 7: The triad significance profiles of 100 graphs grown with random growth rules (SLP weights). Columns are sorted so that the graphs with higher normalised Z-scores for triads with lower ID numbers come first. This highlights that the random search over growth rules has created graphs with a wide range of different significance profiles. Refer to Figure 2 for the key to the 13 triad numbers.

with the results placed into the relevant quadrants via tensor products with the $\mathbf{Q}$ matrices.

The matrix $\mathbf{A}'$ has size $2n \times 2n$ and would be the new adjacency matrix *if* all the nodes were to divide. Since not all the nodes may in fact have chosen to divide, and since some of the existing nodes may have chosen to be removed from the graph, the final step is to reduce $\mathbf{A}'$ to only those nodes that should be kept:

$$\mathbf{A}'' = (\mathbf{A}'_{ij})_{i,j \in \mathbf{r} \cup \mathbf{d}} \tag{8}$$

where $\mathbf{r}$ is a vector of the original node indices that should remain in the graph and $\mathbf{d}$ is a vector of the new node indices of the nodes that have chosen to divide. $\mathbf{A}''$ is the final updated adjacency matrix.

Experiments

Random Search

This experiment aims to establish whether DGCA-M is able to grow graphs with a range of different motif profiles. We randomly initialise the weights of the two SLPs that control the growth process. We use a 3-state system: each node can be in one of three states (in contrast to Figure 4 which illustrates a 2-state system for the sake of simplicity). Using three states rather than two allows a greater range of developmental behaviour (see also Waldegrave et al. (2023b)). Starting with a "seed graph" of a single node, we run the system for up to 256 development steps or until the graph has reached 300 nodes. We discard any graph with < 100 nodes. We also only look at graphs with relatively sparse connectivity, between 0.003 and 0.03, roughly the range of connectivity of the biological networks considered here. We ran this procedure until we had 100 valid graphs, and we then calculated the TSP for each. To get 100 graphs which met these conditions 1632 had to be discarded: 1321 because they were too small (including those where all nodes had been removed), 240 because they were too sparse, and 71 because they were too dense. The results are shown in Figure 7. Each column shows the TSP of one graph over the 13 triads. The columns are sorted to highlight the wide range of different significance profiles found.

This demonstrates that DGCA-M is capable of growing graphs with a range of different motifs and anti-motifs. It appears that, when using random growth rules, it is "easier" for the system to produce graphs with certain triads as motif. For example, in Figure 7 we see 25 of the 100 graphs have the "Mutual-V" triad (number 6) as their most significant motif, whereas only 3 have the "Regulated Mutual" triad (number 9) as their most significant motif.

Goal-directed Evolution

This experiment aims to evolve developmental rules (in the form of SLP weights) that produce graphs with TSPs matching those of various biological networks. This follows the evo-devo loop shown in Figure 1 with the "short circuit" route of judging fitness directly on motif profile rather than on the functional behaviour of the network.

We use the TSPs of two biological networks as targets for the evolutionary search: the *E. Coli* transcriptome and the *C. Elegans* neural network. These are shown in Figure 3. We use one from each "family" of networks. Since $|\mathbf{z}|$ is higher for the *C. Elegans* neural network, indicating that its distribution of triads is more non-random, we might expect that it would be more difficult for evolution to generate a network with this TSP.

We use a modified version of the Microbial Genetic Algorithm (MGA) (Harvey, 2011), to evolve the SLP weights. This is a steady-state evolutionary algorithm in which pairs of individuals are chosen at random to be evaluated. After each contest the losing individual receives some of the genetic material of the winner, inspired by horizontal gene transfer in bacteria. In our case this crossover operation is effected by replacing half of the rows of the losing SLP weights matrix, with the equivalent rows from the winner. The losing individual is also mutated: we randomly change 2% of the SLP weights.

Since we have two separate SLPs (one for action choice and one for state update), we treat the two sets of weights as separate *chromosomes*. We use a population of 6 of each chromosome, combining them to give 36 (6^2) "genetically complete" individuals.

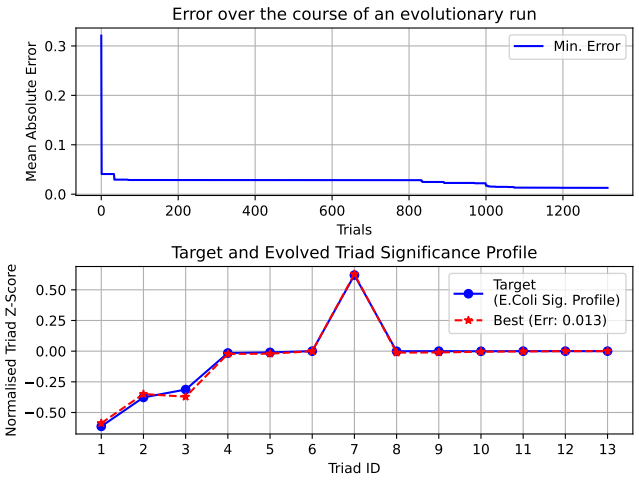

Figure 8: The evolutionary run with the *E.Coli* transcription network TSP as the target. The upper chart shows the minimum MAE over the course of the evolutionary run. The best result was found after 1253 trials, with a MAE of 0.013. The lower chart plots the TSP of the best evolved network against the biological network.

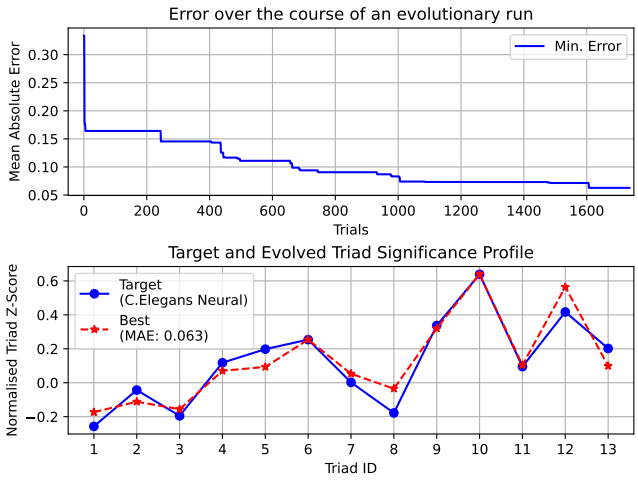

Figure 9: The evolutionary run with the *C.Elegans* neural network TSP as the target. The best result was found after 1607 trials, with a MAE of 0.063.

When two individuals are evaluated, the weaker individual receives genetic material from the stronger in both chromosomes *unless* one of these has previously been part of an individual fitter than the winner. In this case it is not changed since we wish to avoid changing the weights of an SLP that has been effective in combination with a different partner, even if it does not perform well in its current combination.

Since we are running an evo-devo experiment, the fitness evaluation step involves growing the graph. We use the same procedure as described in the random search experiment above.

We also use the same criteria for graph size and range of connectivity. If the grown graph does not meet these conditions, we assign it a fitness of 0 (or an error of ∞), making it automatically the loser of the contest. If it does meet the conditions, we calculate its TSP, as described in the Motif Detection section.

We then calculate the Mean Absolute Error (MAE) of this significance profile compared to the target; the inverse of this error is used as the fitness in the evolutionary process.

The results of running this evolutionary process are shown in Figures 8 and 9. The best evolved growth rules are able to create graphs with MAEs of 0.013 for the *E. Coli* transcription network, and 0.063 for the *C. Elegans* neural network. In the first case the error became low almost immediately: this is probably because it is relatively easy for the system to grow networks with an abundance of triad number 7, as discovered in the random search experiment. It took the evolutionary process longer to find developmental rules to grow a network with a TSP like that of the *C. Elegans* neural network. The random search experiment shows that random rules produce few networks with triad number 10 as their most significant.

Conclusion

This work has built on the DGCA system to allow the growth of graphs with a wide range of TSPs. Used in an evo-devo setup, it can create networks with TSPs similar to those of certain biological networks. These could be useful as ANNs or in an RC context, since it is hypothesised that motifs affect the network's dynamics and function. They could also be useful in other ALife contexts where modular structure is required. Milo et al. (2004) have observed that biological and designed networks can be classified into a few "superfamilies" based on network motifs. The approach introduced here could be used to generate families of networks with similar TSPs to observed biological networks in order to study how much of their behaviour is generically due to the TSP rather than the exact wiring.

Future Work Future work will run the evo-devo process on the full loop shown in Figure 1 and evaluate the developed networks on a range of tasks, rather than assessing fitness based on TSPs.

We also plan to evaluate DGCA-M's ability to create networks with bigger network motifs ($k > 3$) and larger-scale modular structure. In particular, the ability of the system to reproduce network structures whilst also introducing variation suggests it could present an interesting analogue of structures in biological neural networks.

Source Code. Source code for the DGCA-M system and the experiments conducted in this paper is available at: https://github.com/rvrsdl/dgca_motifs

References

Alon, U. (2007). *An Introduction to Systems Biology: Design Principles of Biological Circuits*. Number 10 in Chapman & Hall/CRC Mathematical and Computational Biology Series. Chapman & Hall/CRC, Boca Raton, FL.

Bennett, M. (2020). An Attempt at a Unified Theory of the Neocortical Microcircuit in Sensory Cortex. *Frontiers in Neural Circuits*, 14:40.

Dale, M., Miller, J., Stepney, S., and Trefzer, M. (2019). A substrate-independent framework to characterize reservoir computers. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 475:20180723.

Eom, Y.-H., Lee, S., and Jeong, H. (2006). Exploring local structural organization of metabolic networks using subgraph patterns. *Journal of Theoretical Biology*, 241(4):823–829.

Erdős, P. and Rényi, A. (1959). On random graphs. I. *Publicationes Mathematicae Debrecen*, 6(3-4):290–297.

Grattarola, D., Livi, L., and Alippi, C. (2021). Learning Graph Cellular Automata. In *Advances in Neural Information Processing Systems*, volume 34, pages 20983–20994. Curran Associates, Inc.

Harvey, I. (2011). The Microbial Genetic Algorithm. In Kampis, G., Karsai, I., and Szathmáry, E., editors, *Advances in Artificial Life. Darwin Meets von Neumann*, volume 5778, pages 126–133. Springer Berlin Heidelberg, Berlin, Heidelberg.

Hiesinger, P. R. (2021). *The Self-Assembling Brain*.

Hill, M., Stepney, S., and Wan, F. (2005). Penrose Life: ash and oscillators. In Capcarrere, M. S., Freitas, A. A., Bentley, P. J., Johnson, C. G., and Timmis, J., editors, *ECAL 2005, Canterbury, UK, 2005*, volume 3630 of *LNAI*, pages 471–480. Springer.

Inselberg, A. (1985). The plane with parallel coordinates. *The Visual Computer*, 1(2):69–91.

Jaeger, H. (2001). The" echo state" approach to analysing and training recurrent neural networks-with an erratum note'. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148.

Kashtan, N. and Alon, U. (2005). Spontaneous evolution of modularity and network motifs. *Proceedings of the National Academy of Sciences*, 102(39):13773–13778.

Leier, A., Kuo, P. D., and Banzhaf, W. (2007). Analysis of preferential network motif generation in an artificial regulatory network model created by duplication and divergence. *Advances in Complex Systems*, 10(02):155–172.

Lones, M. A., Turner, A. P., Fuente, L. A., Stepney, S., Caves, L. S. D., and Tyrrell, A. M. (2013). Biochemical connectionism. *Natural Computing*, 12(4):453–472.

Luo, L. (2016). *Principles of Neurobiology*. Garland Science Taylor & Francis Group, New York London.

Marr, C. and Huett, M.-T. (2009). Outer-totalistic cellular automata on graphs. *Physics Letters A*, 373(5):546–549.

Milo, R., Itzkovitz, S., Kashtan, N., Levitt, R., Shen-Orr, S., Ayzenshtat, I., Sheffer, M., and Alon, U. (2004). Superfamilies of Evolved and Designed Networks. *Science*, 303(5663):1538–1542.

Milo, R., Shen-Orr, S., Itzkovitz, S., Kashtan, N., Chklovskii, D., and Alon, U. (2002). Network Motifs: Simple Building Blocks of Complex Networks. *Science*, 298(5594):824–827.

Najarro, E., Sudhakaran, S., and Risi, S. (2023). Towards Self-Assembling Artificial Neural Networks through Neural Developmental Programs. In *ALIFE 2023: Ghost in the Machine: Proceedings of the 2023 Artificial Life Conference*. MIT Press.

Peixoto, T. P. (2014). The graph-tool python library. *figshare*.

Prill, R. J., Iglesias, P. A., and Levchenko, A. (2005). Dynamic Properties of Network Motifs Contribute to Biological Network Organization. *PLOS Biology*, 3(11):e343.

Ryan, K., Lu, Z., and Meinertzhagen, I. A. (2016). The CNS connectome of a tadpole larva of Ciona intestinalis (L.) highlights sidedness in the brain of a chordate sibling. *eLife*, 5:e16962.

Shellman, E. R., Chen, Y., Lin, X., Burant, C. F., and Schnell, S. (2014). Metabolic network motifs can provide novel insights into evolution: The evolutionary origin of Eukaryotic organelles as a case study. *Computational biology and chemistry*, 53PB:242–250.

Stanley, K. O. and Miikkulainen, R. (2002). Evolving Neural Networks through Augmenting Topologies. *Evolutionary Computation*, 10(2):99–127.

Waldegrave, R., Stepney, S., and Trefzer, M. A. (2023a). Developmental Graph Cellular Automata. In *ALIFE 2023: Ghost in the Machine: Proceedings of the 2023 Artificial Life Conference*. MIT Press.

Waldegrave, R., Stepney, S., and Trefzer, M. A. (2023b). Exploring the Rich Behaviour of Developmental Graph Cellular Automata. In *ALIFE 2023: Ghost in the Machine: Proceedings of the 2023 Artificial Life Conference*. MIT Press.

Walter, A., Wu, S., Tyrrell, A. M., McDaid, L., McElholm, M., Sumithran, N. T., Harkin, J., and Trefzer, M. A. (2023). Artificial Neural Microcircuits for use in Neuromorphic System Design. In *ALIFE 2023: Ghost in the Machine: Proceedings of the 2023 Artificial Life Conference*. MIT Press.

Wernicke, S. (2006). Efficient Detection of Network Motifs. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(4):347–359.

White, J. G., Southgate, E., Thomson, J. N., and Brenner, S. (1997). The structure of the nervous system of the nematode Caenorhabditis elegans. *Philosophical Transactions of the Royal Society of London. B, Biological Sciences*, 314(1165):1–340.

Wringe, C., Stepney, S., and Trefzer, M. A. (2023). Modelling and Evaluating Restricted ESNs. In Genova, D. and Kari, J., editors, *Unconventional Computation and Natural Computation*, Lecture Notes in Computer Science, pages 186–201, Cham. Springer Nature Switzerland.

Zhao, L., Beverlin, B., Netoff, T., and Nykamp, D. (2011). Synchronization from Second Order Network Connectivity Statistics. *Frontiers in Computational Neuroscience*, 5.