Imperial College London

BEng Individual Project Interim Report

Imperial College London

Department of Computing

Regenerating Graph Cellular Automata

Author:
Manuj Mishra

Supervisor:
Prof. Andrew Davison

Second Marker:
TBC

January 26, 2022

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

Self-organisation is a remarkable phenomenon in which basic rules and feedback loops between a set of simple, unintelligent agents can lead to the emergence of complex, intelligent-looking behaviour. Numerous examples of self-organisation have been noted in nature from colony-building ants [7] to flocking starlings [8]. Morphogenesis is one form of self-organising behaviour in which agents can self-assemble into highly complex shapes. This can be observed in the cells of most multicellular organisms which develop into specialized tissues, organs, and biological systems. Another property of interest is robustness. When damaged, all organisms have the ability to heal on some level, usually taking on a shape very similar to the original. Certain metazoans, like the planarian flatworm are considered "immortal under the edge of a knife" [9] because a slice of tissue less than 0.5% of the original organism is capable of regrowing into an entirely new self-sufficient individual. The interplay of inherent genomic information and external environmental signals in dictating a cell's actions during morphogenesis and regeneration is an active area of research [10] and understanding these mechanisms is a crucial challenge in the fields of regenerative medicine and bioengineering.

Building computational models of this behaviour can help us understand the processes that generate and sustain self-organising systems in nature. With this knowledge, we can hope to capture the potential of highly distributed self-organising systems in areas such as swarm robotics and computer networks. Cellular automata (CA) are discrete computational models originally studied by von Neumann in the 1940s. They consist of cells using simplistic rules to interact in a grid-like structure. Despite their simplicity, they can exhibit self-organising features and have been widely studied in the field of artificial life.

There has been a long history of work leveraging artificial neural networks to learn and simulate the behaviour of various classes of CA. With the surge of interest in graph neural networks over the last decade, very recent work has started to extend these techniques to a more general class of CA in which the cells no longer have to exist in a regular lattice structure. These generalised CA are called Graph Cellular Automata (GCA).

With this in mind, we aim to "grow" graph cellular automata, leveraging graph neural networks to learn the rules that will allow a GCA point cloud of arbitrary size to approximate a given shape and maintain that shape when subject to perturbation. The key technical challenges throughout this work are as follows:

1. Develop a novel machine learning scheme to ensure training done on point clouds of a given size can be extrapolated to smaller point clouds.
2. Develop a damage-based training pipeline to allow GCAs to learn regenerative behaviour
3. Investigate theoretical methods to increase the probability of convergence to fixed point solutions rather than long-period attractors. This is explained further in Section 3.4.
4. Test these approaches on different point clouds and produce interactive web demonstrations.

Chapter 2

Preliminaries

A cellular automaton (CA) is a regular lattice of computational units called cells. Each cell v is characterised by a state variable $S_i(t) \in \Sigma$, where i indicates the position of the cell in the lattice, t indicates the time, and Σ denotes the (possibly infinite) set of all state variables. Each cell also has a finite local neighbourhood set $\mathcal{N}(v)$ with cardinality N. Typically, the neighbourhood of a cell contains the cell itself (i.e. $v \in \mathcal{N}(v)$). At each time step, the state of every cell is simultaneously updated according to a fixed transition rule; $\phi : \Sigma^N \to \Sigma$ which takes the neighbour states as input. This rule is typically shared across all cells in the CA.

There choice of neighbourhood function is largely directed by the choice in lattice structure. The primary considerations when choosing a neighbour function are the distance metric used to measure proximity of cells and the value of this metric used to define the neighbourhood boundary. There are two very common neighbourhoods used on Euclidean lattices. The *von Neumann neighbourhood* contains all cells within a Manhattan distance of 1. For a 2D square lattice, this contains the cell itself and the 4 cells in the cardinal directions. For a 3D cubic lattice, it contains the central cell and a 6-cell octahedron around it. The *Moore neighbourhood* contains all cells at a Chebyshev distance of 1. For a 2D square lattice, this is the central cell with the 8 neighbouring cells in a square around it. In the 3D case, it is a cube. These are shown in figure 2.1

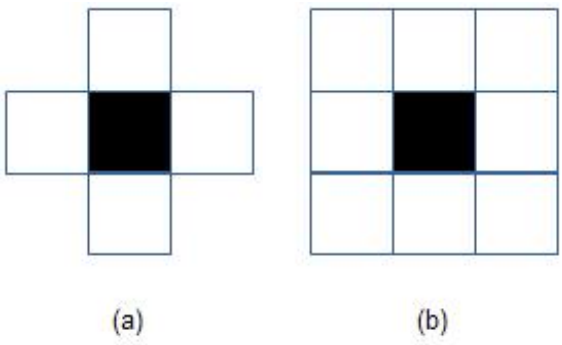


Figure 2.1: (a) von Neumann Neighbourhood and (b) Moore neighbourhood on a 2D square lattice [1]

2.1 Conway's Game of Life

The most popular example of a CA is the Game of Life (henceforth "Life") formulated by John Conway in 1970 [11]. It consists of a 2D grid of cells, each with a boolean state variable signifying that the cell is either "alive" or "dead". The transition rule is a function of the cell's own state $S_i(t)$ and number of living individuals in the cell's Moore neighbourhood (excluding itself), denoted n. This is as follows:

$$\phi(S_i(t), n) = \begin{cases} 0 & S_i(t) = 1 \text{ and } n < 2 \text{ (Death by "loneliness")} \\ 0 & S_i(t) = 1 \text{ and } n > 3 \text{ (Death by "overcrowding")} \\ 1 & S_i(t) = 1 \text{ and } n \in \{2, 3\} \text{ (Survival)} \\ 1 & S_i(t) = 0 \text{ and } n = 3 \text{ (Resurrection)} \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

Despite its simplicity, the Life is Turing complete. There has been a lot of experimentation with Life to discover and classify patterns that can exist within it. Some examples include periodic oscillators like the *beacon* which has period 2 or still lifes like the *block* which are fixed-point solutions 2.2. These can alternatively be thought of as having period 1. There are also periodic patterns which move across the lattice such as the *glider* pattern.

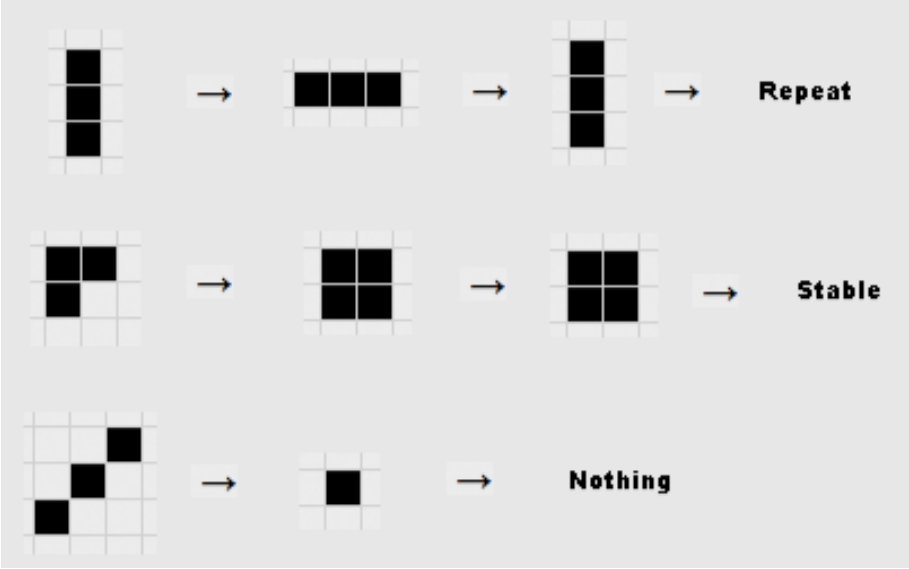


Figure 2.2: Patterns within the Game of Life. From top to bottom: blinker, block, unnamed [2]

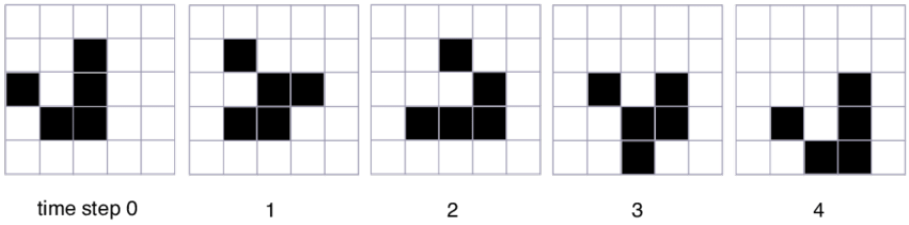


Figure 2.3: The glider pattern within the Game of Life [3]

2.2 Wolfram's Classification

The choice of lattice geometry, neighbourhood function, state variable, and transition rule define the behaviour of a CA given some initial condition. Fixing the former three factors, Wolfram [12] has classified CAs based on transition rules as follows:

1. Class 1 (Null) : Rules that lead to a trivial, uniform state
2. Class 2 (Fixed point + periodic) : Rules that lead to stable or periodic patterns
3. Class 3 (Chaotic) : Rules that lead to chaotic patterns
4. Class 4 (Complex) : Rules that lead to complex, long-lived impermanent patterns

Elementary cellular automata are defined on the simplest nontrivial lattice, a finite one-dimensional chain. The neighbourhood of each cell contains the cell itself and the two cells adjacent to it on either side. The state variable is a boolean which means there are $2^3 = 8$ possible neighbourhood state configurations. A transition rule maps each of these neighbourhood states to a resultant state and can therefore be represented as an 8-digit binary rule table $(t_7t_6t_5t_4t_3t_2t_1t_0)$ where configuration (000) maps to t_0, (001) maps to t_1, ..., and (111) maps to (t_7). Consequently there are $2^8 = 256$ possible transition functions for elementary CA. Figure 2.4 illustrates the distribution of classes in this rule space as we vary p, the proportion of 1s in the rule table.

The Wolfram code, a number between 0 and 255 obtained by converting the binary rule table to decimal, is the standard naming convention for these rules. Rule 110 is particularly notable as it can exhibit class 4 behaviour [5] and has been proven to be Turing Complete [13]. Figure 2.5 shows an example progression of a Rule 110 system. Each row of pixels represents the state of the automaton at one snapshot in time with the topmost row representing the randomized initial state. It shows the emergence, interaction, and subsequent dissipation of multiple long-lived impermanent patterns.

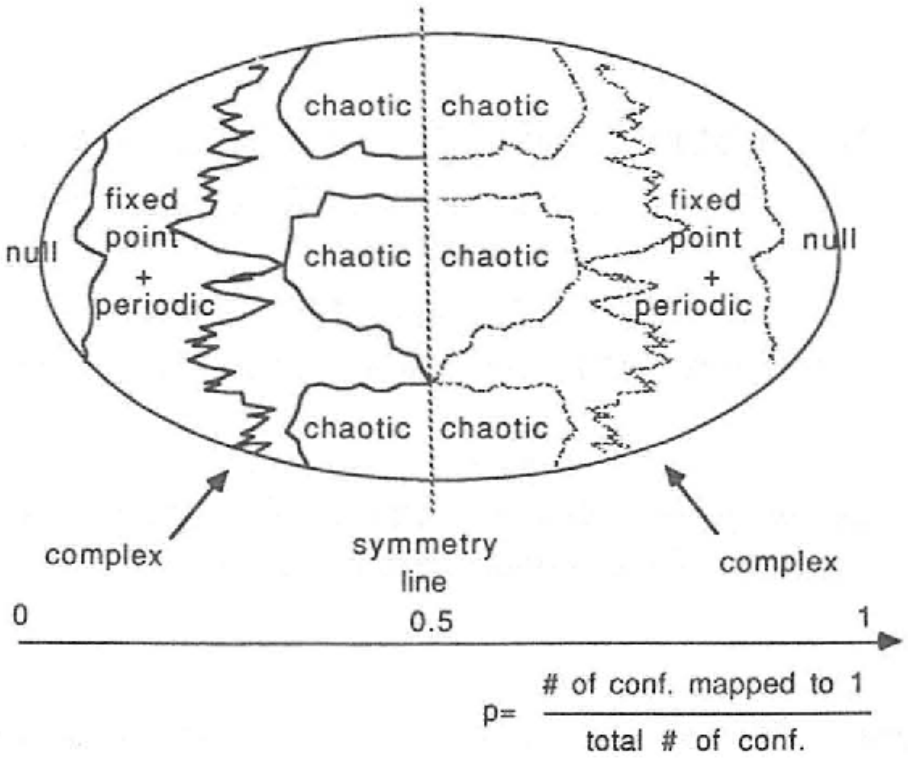


Figure 2.4: Schematic illustration of elementary CA rule space from Li and Packard [4]

2.3 Morphogenesis

Morphogenesis is the process by which a system develops into a particular shape or pattern. Biologically, this is seen in most multicellular organisms which can robustly develop specialised organs and intricate skin patterns without any centralised decision-making. Through simple rules encoded in the genome and homeostatic feedback loops enforced through chemical signalling, a tissue knows exactly how to grow and when to stop.

Emulating this behaviour *in silico* can provide great insight into the way self-organising and self-repairing biological systems function. Cellular automata are a promising model of computation for artificial life simulation because, much like biological agents, their behaviour follows logically

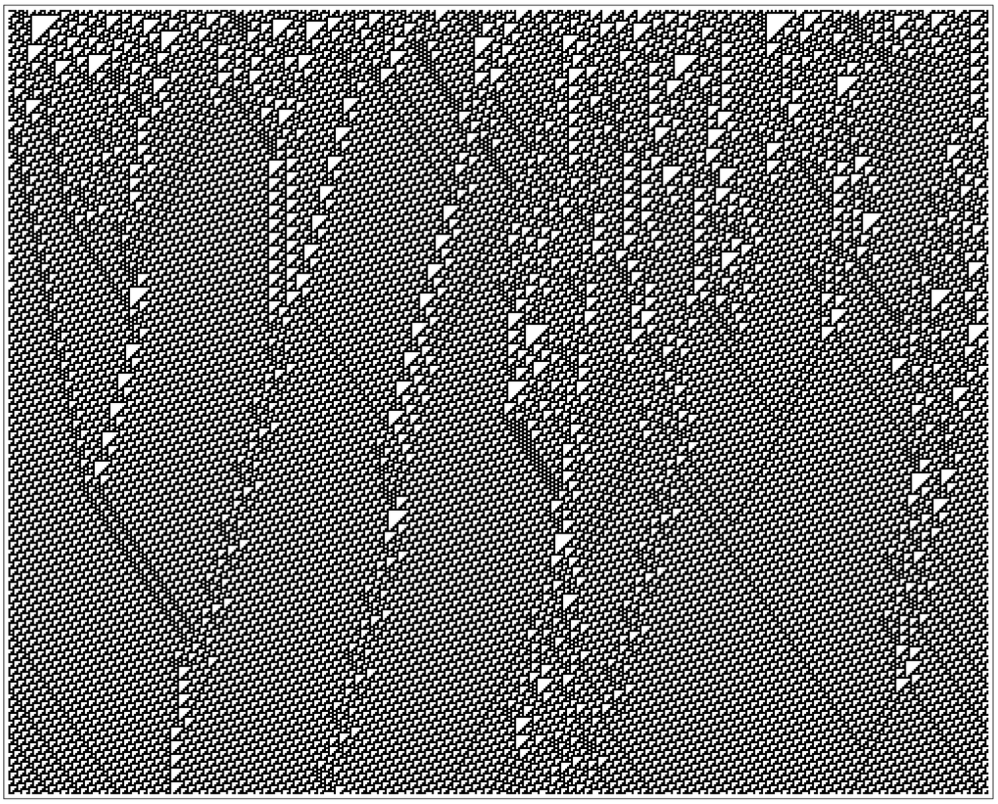


Figure 2.5: Rule 110 progression with random initialisation [5]

from combining information in their surroundings and internal programming. In the context of morphogenesis, we are interested in rules that form stable class 2 patterns from random initial conditions. We are also interested in rules that are resistant to noise. This is analagous to the behaviour of biological cells which grow into stable configurations and are robust to pertubation and damage during growth.

Chapter 3

Related Works

3.1 Learning cellular automaton dynamics

Learning Cellular Automaton Dynamics with Neural Networks (Wulff and Hertz, 1992) [14] is a seminal work in this area. It uses a lattice-shaped network with the same structure as the CA being simulated. Each node in this lattice is a Probabilistic Logic Node, also known as a $\Sigma - \Pi$ unit [15]. These units are capable of representing any boolean function. That is to say $\forall f : \mathbb{R}^N \to \{-1, 1\}, \exists\ w_1, w_2, ..., w_N \in \mathbb{R}$ such that,

$$f(S_1, S_2, ..., S_N) = sgn\left[\sum_{j=1}^{N} w_j \prod_{i \in I_j} S_i(t)\right] \tag{3.1}$$

where index set I_j is randomly drawn from the integers $\{1, 2, ..., N\}$ without replacement. Notably, these units do not require input from every neighbour to learn successfully. In fact, this paper found any index set of size $|I_j| \geq \frac{N}{2}$ to be sufficient. This insight significantly reduced training time.
3 learning goals were established for the network. In order of increasing difficulty they were:

1. Extrapolation: Learn to simulate a CA for a *particular* initial condition at any time
2. Dynamics : Learn to simulate a CA for *any* initial condition after short-lived patterns have been exhausted
3. Full Rule : Learn to simulate a CA for any initial condition at any time

This work was largely concerned with class 3 (chaotic) and class 4 (complex) behaviour. All 9 known examples of class 3 1D automata were tested on. However, at the time, it was believed that 1D CAs could not exhibit class 4 behaviour so testing was also conducted on Conway's Game of Life. There were two settings under which testing was done.

1. Shared weights: There was a single network learning a single transition rule across the automata.
2. Individual weights: Each cell was learning its own transition rule based on information available from its local neighbourhood only.

With shared weights, this approach was very promising, with extrapolation and dynamics being very easy in the 1D and 2D case. Learning the full rule was much harder with the network only being able to do so for 4 out of the 9 candidates in the 1D case.

With individual weights, all learning was difficult. In the 1D case, extrapolation was still possible for all candidates but dynamics was only possible for a single candidate, rule 22. Learning Life was also impossible with the network failing to extrapolate even when given several hundred steps of history.

As the first notable exploration of learning transition rules with neural networks, this paper demonstrated a method for learning chaotic behaviour in cellular automata. It also divided class 3 elementary CAs into two categories according to how easy it is to learn their underlying rule. Furthermore, it raised many questions for future research. The most pertinent is whether these results on a few simple examples generalise to all complex and chaotic CAs.

However, this work only explored class 3 and 4 CAs, presumably because these are the most interesting varieties. For the goal of morphogenesis, we are more interested in the possibility of learning on class 2 CAs.

3.2 Using evolutionary algorithms

Another interesting line of research is the use of evolutionary algorithms to evolve ANNs for CA morphogenesis.

Evolving Self-organizing Cellular Automata Based on Neural Network Genotypes (Elmenreich and Fehérvári, 2011) [6] is an early work in this area. Each cell in a CA is controlled with an ANN with 9 input nodes, a 6-node hidden layer, and 5 output nodes. One output node indicates the cell's colour while the others propagate information to the cell's 4-neighbourhood. The input nodes take in the corresponding information from the cell's 4-neighbourhood as well as the colour of all 4 neighbours. The last input node takes in the current colour of the cell itself. Although each cell is instantiated with the same ANN, the internal state of each ANN can adapt independently at each time step. The evolution procedure begins by initialising a population of 100 candidate ANNs with random weights and applying algorithm 1 repeatedly.

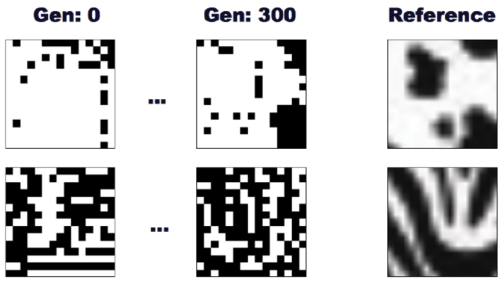
Algorithm 1 Evolutionary Algorithm to improve ANNs

for all generations **do**
  for $i = 0$ to 100 **do**
    Evaluate network i and store score
  end for
  Rank networks and create new population as follows:
  Elitism: Select top 15 candidates
  Selection: Randomly select 10 from remainder with a small bias towards fitness and diversity
  Mutations: Copy a randomly selected candidate and mutate its weights and biases. Repeatedly apply this operation until there is a new set of 30 mutated candidates.
  Recombinations: Copy a randomly selected pair of candidates and apply crossover. Repeatedly apply this operation until there is a new set of 40 recombined candidates.
  Concatenate mutated and recombined candidates with the original selected candidates.
  Reinitalise: Initialise 5 new networks with random weights and add to new population
end for

Although this work was able to successfully learn simple pattern like flags (3.1), it failed to learn on more complicated structures like animal skin patterns (3.2a) or the Mona Lisa (3.2b). This is likely due to a poor choice of fitness function given the task at hand. By comparing pixel-by-pixel, candidates with a largely similar pattern to the reference image in a qualitative sense are scored poorly compared to those that can precisely imitate very small portions of the image.

Another problem with evolutionary algorithms in general is that they get stuck in suboptimal fitness maxima. Finally, the broad learning network architecture is flawed because only border cells can infer information about their position in the image. Since the ratio of inner to outer cells increases for higher resolution images, it gets increasingly difficult to pass this information to central cells. Therefore this solution is likely to scale poorly as was seen in the 20x29 Mona Lisa case.

Figure 3.1: Successful reproduction of the the Austrian Flag over 90 generations [6]



(a) Attempt at learning cow and zebra skin patterns over 300 generations [6]

(b) Attempt at learning the Mona Lisa over 500 generations. Depicted from left to right are the original, the reference, and the best reproduction attempt [6]

Figure 3.2: Unsuccessful attempts to learn complex patterns over 300+ generations

CA-NEAT: Evolved Compositional Pattern Producing Networks for Cellular Automata Morphogenesis and Replication (Nichele et al., 2018) [16] elicited more promising results by opting for a different genotype and evolution strategy. The genotypes here were compositional pattern producing networks (CPPNs) which were evolved through a neuroevolution of augmenting topologies (NEAT) algorithm. A CPPN is a type of ANN that is particularly well suited to evolution through genetic algorithms. Unlike typical ANNs, these networks do not have uniform activation functions across layers. Instead, a variety of different activation functions are chosen to induce specific patterns in the output. For example, Gaussian activations are used to create symmetricity and sinusoidal activations are used to create repetition.

NEAT is a genetic algorithm designed especially for ANN evolution [17]. The initial population consists of simple networks with no hidden layers. Over multiple generations, nodes and connections are added or disabled. Activation functions and weights are also adjusted. As individuals become increasingly dissimilar, a compatibility distance metric is used to segregate individuals into seperate species once they cross a threshhold of incompatibility. Pairs for reproduction are only selected within species. New species are granted a period of immunity during which they won't be eliminated by the algorithm but subsequent poor performance does lead to the removal of failing species to ensure continuous improvement in the fitness of the population.

The two goals established by this paper were morphogenesis - to develop a given pattern from a simple seed, and reproduction - to produce at least 3 copies of a pattern from a single instance. For the purpose of this review we will focus on the morphogenesis goal. The fitness function for this is given in equation 3.2

$$f(x) = xe^{5(x-1)} \tag{3.2}$$

$$\text{where } x = \max_{t=1..30}(r_t) \tag{3.3}$$

where r_t is the ratio of cells with the correct state at time t. This exponential fitness function suppresses the contribution of dormant cells (i.e. low values of x) while maintaining the essential invariant $f(1) = 1$. This choice biases the algorithm towards more active CA.

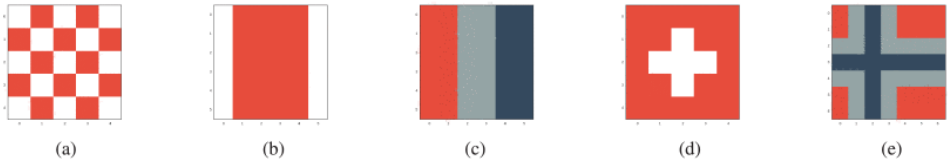


Figure 3.3: Five goal patterns for CA-NEAT

(a) *Mosaic*, (b) *Border*, (c) *Tricolor*, (d) *Swiss*, (e) *Nordic*

This method was tested on 5 simplistic patterns and proved to have improvements over existing techniques like table-based evolution and instruction-based evolution [18] for certain structures like the *Mosaic* and *Tricolor* patterns while performing worse for others like the *Border* pattern.

In order to further improve the model, it would be useful to include a bias against complexity in the topology of the CPPNs because simpler models tend to generalise better to unseen scenarios. It would also be useful to penalise the length of cycles of periodic attractors to incentivise the model to find fixed-point solutions. Finally, a novelty search approach may be an interesting avenue of exploration to avoid getting trapped in local maxima.

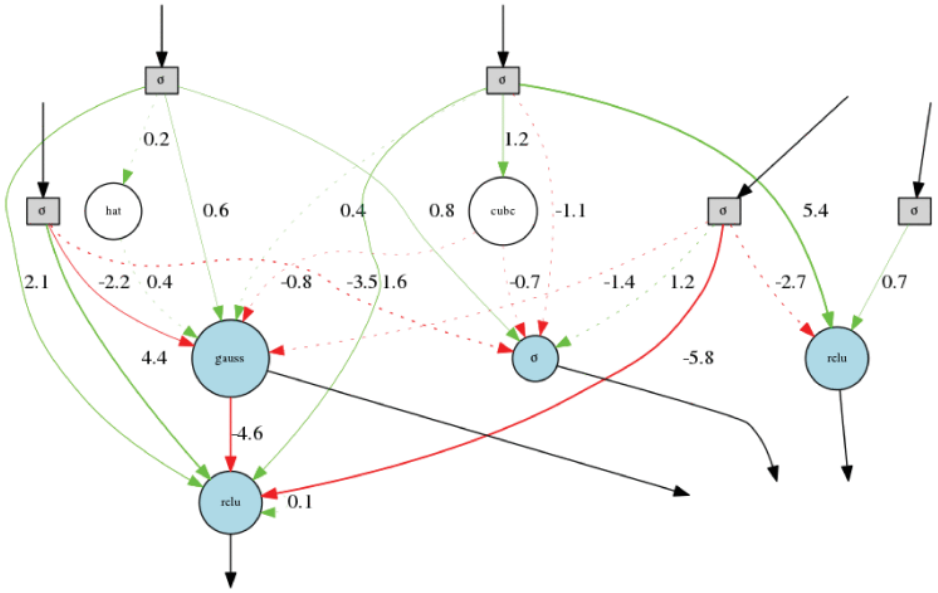


Figure 3.4: Fixed point CPPN solution found for *Tricolor* morphogenesis

Dotted lines indicate disabled connections. Note the two vestigial hidden layer nodes

3.3 Using convolutional neural networks

Growing Neural Cellular Automata (Mordvintsev et al., 2020) [19] is a recent paper showing how CNNs can be applied to the problem of morphogenesis, producing stable complex image patterns from a single seed state. These structures are made extremely robust to perturbation through damage-based training.

A 2D CA model with a Moore neighbourhood and continuous vector state variable is used. The continuous state allows the system to learn a differentiable update rule which can be optimized

through a convolutional neural network. The model architecture defines a state with 12 hidden channels and 4 visible channels. The visible channels include the 3 RGB values and a special α channel which represents the vitality of a cell. In particular there are 3 types of cell. Mature cells have $\alpha > 0.1$. Together with their neighbours, these cells are considered "living". Neighbours of mature cells with $\alpha \leq 0.1$ are considered "growing". All other cells are considered "dead" and have their state vector reset to $\underline{\mathbf{0}}$ at each time step. The hidden vectors can be thought of as an opaque encoding mechanism in the cell, much like chemical concentration or electric potentials in biological cells.

The training pipeline features 4 key steps. The first is *perception* in which a 3x3 convolution is applied. Two classical Sobel filters are used estimate the partial derivative of the state vectors in the $\overrightarrow{x}$ and $\overrightarrow{y}$ directions. These represent the cells' signalling mechanism. The biological analogy here would be chemical gradients or electrical signal pathways. The gradients are flattened into a vector which is fed into a neural network of multiple layers including 1x1 convolutions and ReLU activations. The result is iteratively fed through this network for a random number of steps in the range $[64, 96]$. At each update, a per-cell stochastic dropout is applied to simulate the a random time-interval between cell updates. This is to avoid the possibly misleading assumption of global synchronisation between cells. The pixel-wise Euclidean loss between the RGBA channels is then optimized through backpropagation-through-time. The resultant networks from this training represent candidate update rules that can induce the growth of a pattern from a single seed.

However, these networks exhibited drastic instability in the long term. To produce persistent patterns, a sample pool strategy is used whereby the network is trained on a pool of automata simultaneously. At each step, a batch of this pool are iterated using the network. The individual with the highest loss in the batch is reset to the seed state to prevent catastrophic forgetting. Using this method, the network learns to recover from incomplete and incorrect states which makes it more likely to converge towards a persistent solution. A simpler option to acheive persistence would've been to let the model train for longer and periodically apply a loss with exponential decaying intervals between these applications. However, this would have drastically increased memory and training time requirements.

One surprising finding of this paper was that some of the resultant models exhibited regenerative behaviour when damaged, despite not being explicitly trained to do so. This indicates a strong overlap between solutions that exhibit persistance and those that exhibit regenerative properties.

In order to further explore these regenerative properties, a new experiment was conducted in which a few samples in each batch were damaged before each training step. The system is therefore trained to recover from half-formed states. This proved very successful. The models produced were able to grow, persist, and recover from different types of damage.

[PUT CRITICISM HERE]

3.4 Using graph neural networks

Learning Graph Cellular Automata (Grattarola et al., 2021) [20] is a recent work introducing the concept of Graph Neural Cellular Automata (GNCA). This structure extends the concept of neural cellular automata to a generalised version of CA called Graph Cellular Automata (GCA) whereby the lattice structure is replaced with an arbitrary graph. Graph neural networks are used to learn transition rules on these GCA.

This paper uses two multilayer preceptrons (MLPs) connected with a message passing layer to represent the transition function for a generalised GCA. The powerful capabilities of this simple architecture are exhibited in 3 experiments of increasing difficulty. They are:

1. Voronoi: Learn an outer-totalistic rule that flips the binary state of a cell depending on the density of neighbours. This is analagous to Conway's Game of Life except on a Voronoi tessellation instead of a regular lattice. This is fairly easy since there is a explicit, known target rule and the graph topology is static.

2. Boids : Learn Reynold's algorithm [21] to simulate the flocking of birds. This GCNA learns on a multi-agent system of points. The state of each point is a multidimensional continuous vector containing the position and velocity of each point as "visible" characteristics. This problem is more difficult. Although the target transition rule is still known, it is far more complex than the Voronoi experiment. Moreover, the graph topology is constantly changing as the points move.

3. 3D Morphogenesis : Learn a rule for a point cloud to converge to a specified shape such that the connectivity of points have some geometrical meaning. The difficulty here arises from the fact that the target rule is unknown. This means we have little information about the attractiveness and periodicity of solutions.

This work was successful at discovering rules that can converge and persist complex target graphs including the Stanford Bunny and Minnesota road network . The key problem with the approach is that the GNCA sometimes converges to periodic solutions that orbit around the target state. Both fixed point solutions and periodic solutions are class 2 CA under the Wolfram classification. We can speculate that the model converges to periodic solutions at times because they are "close" to fixed point solutions in the rule space.

cite

cite

put smth here

Chapter 4

Project Plan

This project is based on improving and building upon very recent work in a relatively new line of exploration. As such, much of the work is experimental. This project demands a level of familiarity with the relevant theory, methods, and technologies around CA and machine learning which is developed in the early stages. The middle stages begin by reproducing existing results from recent work on CA morphogenesis and regeneration. This includes new research into morphogenesis for graph CA. Following this, the key goal is to developing theoretical ideas to improve the methods used to produce these results. These could be more efficient versions of existing ideas, extrapolating ideas from one domain to another (e.g. applying pattern-damage training methods from Mordvintsev et al [19] to the context of graph CA introduced by Grattarola et al. [20]), or even entirely novel ideas. Finally, the project aims to test and improve the efficacy of these methods through experimentation and iteration. At this stage, extension goals could also be addressed. These include practical work like producing rich demonstrations to allow layman audiences to interact with the research. They could also include theoretical work like investigating the connection between periodic and fixed-point solutions in CA rule space and devising a way to incentivise a system to converge to a fixed point solution over a periodic one. Each of these 3 stages is expected to take a roughly equal amount of time. Throughout each of the stages, the project write-up will continue to develop. The interim report will be written during the latter half of stage 1 and the earlier half of stage 2. This will provide a strong basis for the final report which will be written in the latter half of stage 2 and the earlier half of stage 3.

4.1 Key Milestones

The key milestones in this project along with the projected dates of completion are outlined as follows.

Nov / Dec 2021: The first milestone is to decide on specific aims for the project. This was achieved by the planned deadline. The goals for this project came naturally out of an exploration into the literature around morphogenesis in cellular automata. Since CA are a relatively mature concept, there is a large body of research in this area. However, the background reading for this project focused on a subset of this body, concerning the applications of machine learning to learn transition rules. Since the seminal paper by Wulff and Hertz in 1992 [14], this was a relatively stagnant area of research until the breakthrough work by Mordvintsev et al in 2020 [19]. This opened up many possible areas of exploration including self-classifying CAs [22], adversarial attacks on morphogenetic CAs [23], and learning graph CA [20].

Jan / Feb 2022: Following this, the next goal is to write an interim report which introduces the topic, lays out preliminary knowledge, discusses related work, and details a plan for project execution and evaluation. It also lays a strong basis for the final project report. This goal was also achieved on time. The next technical milestone is to replicate the procedure outlined in the Learning Graph Cellular Automata paper [20] and reproduce the results on the 5 example point clouds. It also involves experimenting on new point clouds, eliciting both periodic and fixed-point behaviour. This will allow me to deeply familiarise myself with the methods used in this paper and gain an intuition for ways in which they could be improved.

Mar / Apr 2022: The next milestone would be to devise appropriate machine learning models to train a GCA to perform morphogenesis on point clouds that are smaller in number than the target. This will involve going beyond current research on GCA to develop a better understanding of shape and structure. For example, the GCA will need to understand what it means for two point clouds of differing density to both be in "the shape of a bunny". This will require a more complex metric than the simple Euclidean distance between training points and target points that was used when training and testing on point clouds of equal density. This training method should then be adapted to induce regenerative behaviour in solutions. One promising avenue of exploration here would be the damage-based training techniques seen in Growing Cellular Automata [19].

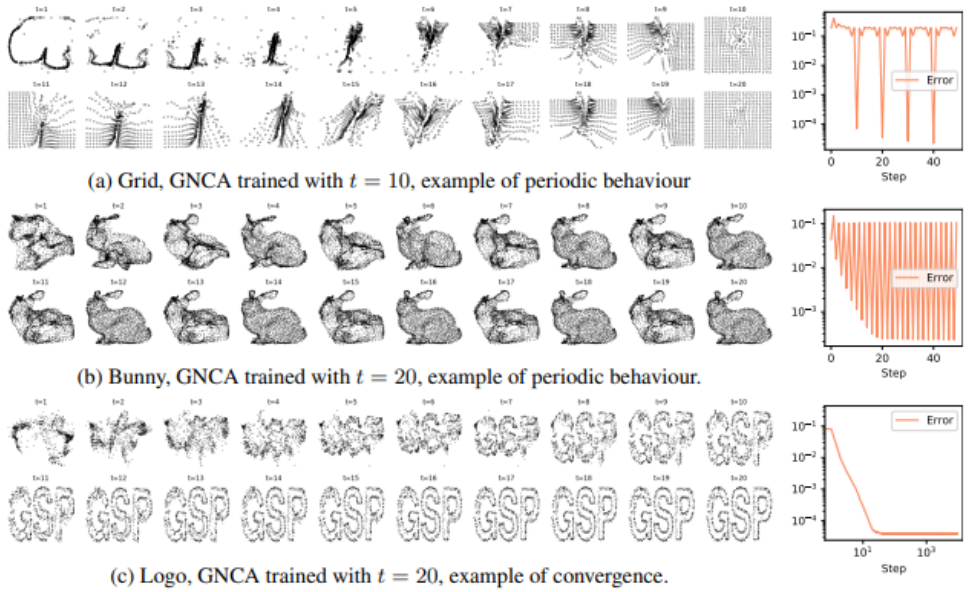
May / Jun 2022: At this point, the goals for the final two months of this project are flexible. If the project is progressing slower than planned, then these months provide a good opportunity to catch up. In the event of a pivot, these months also provide some safety buffer to expand the scope of the project in a different direction from a fall-back position. For example, if the damage-based training proves to be inadequate at producing regenerating solutions in the graph context, then exploring new techniques from different areas of research and developing new ideas will require some additional time that these months can provide. However, if the project is progressing well, then these months will be used to engage with extension goals. One extension goal is to produce explorable explanations of the research. This will involve rewriting some of the machine learning models in Tensorflow.js and producing interactive web demonstrations in a style similar to that used by the Distill journal [24]. Another extension goal is to explore the theory behind different types of class 2 CAs. This is of particular interest as it is useful to seperate fixed-point solutions from periodic solutions and understand the difference between them. The ideal scenario here would be to devise a method of incentivising machine learning systems to converge to fixed-point solutions over periodic solutions. Although the final report will continue to develop throughout the whole project, a substantial portion of the last month will also be dedicated towards improving and polishing the final report.

Chapter 5

Evaluation Plan

The evaluation of this project will be both qualitative and quantitative. Ultimately, deciding whether something is in the "correct shape" is a subjective task and will consequently require some degree of human judgement. Therefore, it will quite easy for us to intuitively judge whether this project has succeeded in producing morphogenetic graph cellular automata that can retain and regenerate their shape when subject to perturbation or damage. Within this analysis, there are many factors we could vary. For example, we could vary the degree of perturbation, the number of points being perturbed, or the shape / size of damage. On the other hand, we could keep these factors constant and vary the density of the point cloud. In this case, we would test the algorithm on many point clouds where each one has fewer points than the previous. The relevant qualitative question here would be: at which point does the point cloud no longer resemble the original shape that was used for training?

However, there are also quantitative components to the evaluation of this project. One key question to ask is how quickly the GNCA converges to a solution and how often it converges to a periodic solution instead of a fixed-point solution. The former question can be compared against the Learning Graph Cellular Automata [20] which provides many useful benchmarks. A selection of these are shown in figure 5.1.



(a) Grid, GNCA trained with $t = 10$, example of periodic behaviour

(b) Bunny, GNCA trained with $t = 20$, example of periodic behaviour.

(c) Logo, GNCA trained with $t = 20$, example of convergence.

Figure 5.1: State of each point cloud between $t = 10$ and $t = 20$ alongside mean squared error between current and target state

Bibliography

[1] Debasis Das. A survey on cellular automata and its applications. volume 269, 12 2011. ISBN 978-3-642-29218-7. doi: 10.1007/978-3-642-29219-4_84.

[2] Chris Lipa. Conway's game of life. URL http://pi.math.cornell.edu/~lipa/mec/lesson6.html.

[3] Alan Dorin, Jonathan McCabe, Jon McCormack, Gordon Monro, and Mitchell Whitelaw. A framework for understanding generative art. *Digital Creativity*, 23, 12 2012. doi: 10.1080/14626268.2012.709940.

[4] Wentian Li, Norman Packard, et al. The structure of the elementary cellular automata rule space. *Complex systems*, 4(3):281–297, 1990.

[5] Stephen Wolfram. *A New Kind of Science*. Wolfram Media, 2002. ISBN 1579550088. URL https://www.wolframscience.com.

[6] Wilfried Elmenreich and István Fehérvári. Evolving self-organizing cellular automata based on neural network genotypes. In *International Workshop on Self-Organizing Systems*, pages 16–25. Springer, 2011.

[7] Eric Bonabeau, Guy Theraulaz, Jean-Louls Deneubourg, Serge Aron, and Scott Camazine. Self-organization in social insects. *Trends in ecology & evolution*, 12(5):188–193, 1997.

[8] Iain D Couzin, Jens Krause, et al. Self-organization and collective behavior in vertebrates. *Advances in the Study of Behavior*, 32(1):10–1016, 2003.

[9] John Graham Dalyell. *Observations on some interesting phenomena in animal physiology, exhibited by several species of Planariae : illustrated by coloured figures of living animals / by John Graham Dalyell.* Edinburgh :Archibald Constable,, 1814. URL https://www.biodiversitylibrary.org/item/40487. https://www.biodiversitylibrary.org/bibliography/10135.

[10] Vaibhav P Pai, Sherry Aw, Tal Shomrat, Joan M Lemire, and Michael Levin. Transmembrane voltage potential controls embryonic eye patterning in xenopus laevis. *Development*, 139(2):313–323, 2012.

[11] Martin Gardner. The fantastic combinations of jhon conway's new solitaire game'life. *Sc. Am.*, 223:20–123, 1970.

[12] Stephen Wolfram. Theory and applications of cellular automata. *World Scientific*, 1986.

[13] Matthew Cook et al. Universality in elementary cellular automata. *Complex systems*, 15(1):1–40, 2004.

[14] N Wulff and J A Hertz. Learning cellular automaton dynamics with neural networks. *Advances in Neural Information Processing Systems*, 5:631–638, 1992.

[15] Kevin N Gurney. Training nets of hardware realizable sigma-pi units. *Neural Networks*, 5(2):289–303, 1992.

[16] Stefano Nichele, Mathias Berild Ose, Sebastian Risi, and Gunnar Tufte. Ca-neat: evolved compositional pattern producing networks for cellular automata morphogenesis and replication. *IEEE Transactions on Cognitive and Developmental Systems*, 10(3):687–700, 2017.

[17] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[18] Stefano Nichele and Gunnar Tufte. Evolutionary growth of genomes for the development and replication of multicellular organisms with indirect encoding. In *2014 IEEE International Conference on Evolvable Systems*, pages 141–148. IEEE, 2014.

[19] Alexander Mordvintsev, Ettore Randazzo, Eyvind Niklasson, and Michael Levin. Growing neural cellular automata. *Distill*, 2020. doi: 10.23915/distill.00023. https://distill.pub/2020/growing-ca.

[20] Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Learning graph cellular automata. *Advances in Neural Information Processing Systems*, 34, 2021.

[21] Craig W Reynolds. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 25–34, 1987.

[22] Ettore Randazzo, Alexander Mordvintsev, Eyvind Niklasson, Michael Levin, and Sam Greydanus. Self-classifying mnist digits. *Distill*, 2020. doi: 10.23915/distill.00027.002. https://distill.pub/2020/selforg/mnist.

[23] Ettore Randazzo, Alexander Mordvintsev, Eyvind Niklasson, and Michael Levin. Adversarial reprogramming of neural cellular automata. *Distill*, 2021. doi: 10.23915/distill.00027.004. https://distill.pub/selforg/2021/adversarial.

[24] Latest articles about machine learning. URL https://distill.pub/.