Imperial College London

BEng Individual Project Final Report

Imperial College London

Department of Computing

Evolutionary Algorithms for the Discovery of Cellular Automata Transition Functions

Author:
Manuj Mishra

Supervisor:
Prof. Andrew Davison

Second Marker:
Dr. Edward Johns

May 16, 2022

Abstract

Evolutionary algorithms are effective tools for black-box optimisation problems.

Acknowledgements

[Acknowledgements here]

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Overview

Evolutionary algorithms (EAs) have long been held as effective tools in black-box optimisation problems. Grounded in the principles of Darwinian evolution, EAs traverse over a search space by performing selection, mutation, and crossover on a population of candidate solutions. Increasingly strong solutions are discovered as the fitness of the population grows.

Cellular automata (CAs) are discrete models of computation in which multiple "cells" are simultaneously updated at regular time steps such that the state of any cell depends exclusively on the state of the cells in a local neighbourhood around it in the previous time step. CAs are powerful computational engines due to their inherently parallel structure. However, they are more commonly utilised as an abstract representation to study nonlinear dynamics and the emergence of complexity.

Top-down investigations into CA behaviour are vast and varied. Mathematical analyses seek to taxonomize CA properties and prove general results about long-term behaviour from intrisic properties. In the natural and social sciences, CAs are designed to model physical, biological, or human behaviour. Both of these endeavours seek to analyse the behaviour of a CA from its structure, transition function, and possibly initial conditions.

In this thesis, we explore a bottom-up approach where we deduce the underlying properties of a CA by observing its behaviour. In particular, we utilise EAs to search the rulespaces of several classes of CA. We tackle multiple objectives from imitation of particular CA behaviour to generation of desirable long-term states. The ability of a rule to compactly encode the future development of a CA is reminiscent of DNA as an encoding mechanism for the development of biological entities. Noting this, it is very apt that we use biologically-inspired algorithms to optimise CAs.

1.2 Contributions

1. We design a genetic algorithm to learn the transition rule of binary outer-totalistic (aka life-like) cellular automata.
2. We use this to build a maze generator that utilises life-like cellular automata to procedurally generate mazes with particular properties based on user preference.
3. We design an evolutionary strategy algorithm to learn to extrapolate and fully simulate Gray-Scott diffusion-reaction equations.
4. We use this to build a network generator that utilises diffusion-reaction continuous automata to procedurally generate efficient networks.

Chapter 2

Preliminaries

2.1 Cellular Automata

A cellular automaton (CA) is a computational model that performs multiple parallel computations, each depending only on local interactions, to produce complex global behaviour. We define a CA formally as follows.

Definition (Cellular automaton)**.** *A cellular automaton is an n-dimensional finite grid of computational units called cells. Each cell c_i is characterised by:*

- *A discrete state variable $\sigma_i(t) \in \Sigma$, where i indicates the index of the cell in the lattice, t indicates the current time step, and Σ denotes the finite set of all state variables.*
- *A finite local neighbourhood set $\mathcal{N}(c_i)$ with cardinality N.*
- *A transition function $\phi : \Sigma^N \to \Sigma$ which takes local neighbour states as input. This is also known as the CA "update rule".*

At each time step, the state of each cell is simultaneously updated according to the transition function. That is, $\sigma_i(t+1) = \phi(\ \{\sigma_j(t) \mid c_j \in \mathcal{N}(c_i)\}\ )$

Due to the breadth of systems studied in CA literature, the constraints of this definition are often altered to produce interesting arrangements. For example:

- The structure need not be a square grid. CA have been studied on hexagonal grids[CITE], aperiodic tessellations such as as the Penrose tiling[CITE], and even randomly generated structures like the Voronoi partition[CITE].
- The system need not be deterministic. Probabalistic cellular automata (PCA)[CITE] have stochastic transition functions which describe a probability distribution of possible outcomes for any given input. PCA are able to model random dynamical systems in the real world from stock markets[CITE] to infectious diseases[CITE].
- The state space Σ need not be finite. In this thesis we will explore multiple possible state variable representations including bit arrays and continuous vectors.

For the purpose of this thesis, we will assume the original definition of CA unless otherwise stated.

2.1.1 Neighbourhood Functions

We consider a "neighbourhood function" for each cell $c_i \mapsto \mathcal{N}(c_i)$. This makes it easier to discuss neighbourhood sets of cells in the CA, each of which are typically homogenous.

There are many possible neighbourhood functions for any given CA geometry. When defining the neighbourhood function, we select a distance metric $d : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ to measure the proximity of two cells and we set a threshold T under which we consider two cells to be within each other's neighbourhood.

$$c_i \in \mathcal{N}(c_j) \iff d(c_i, c_j) \leq T$$

There are two neighbourhoods that are frequently 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.

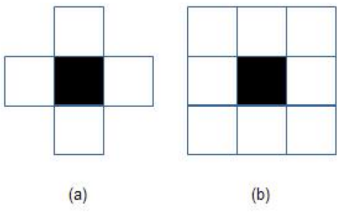


Figure 2.1: (a) von Neumann Neighbourhood and (b) Moore neighbourhood on a 2D square lattice [1]

In a finite grid CA, border cells must be given special consideration since they do not have the same number of neighbours as interior cells and therefore cannot share the same neighbourhood function. One option is to define a case-wise neighbourhood function with different behaviour for border cells. Another option is to freeze the state of border cells. In the field of partial differential equations, this is known as setting "fixed boundary conditions". The problem can also be circumvented entirely by relaxing the finite grid assumption and allowing cells to "wrap around" the grid. This is known as setting "periodic boundary conditions" and can be imagined visually as running the CA on an infinite periodic tiling or, alternatively, on a torus.

2.1.2 Conway's Game of Life

A popular example of a CA is the Game of Life (henceforth "Life") formulated by John Conway in 1970 [7]. 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 takes as input the cell's own state $\sigma_i(t)$ and the number of living individuals in the cell's Moore neighbourhood (excluding itself), denoted n. This is as follows:

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

Despite its simple setup and update rule, Life can exhibit the emergence of complex patterns. It is possible to simulate a fully universal Turing machine within Life [CITE] and, as a corollary of the Halting Problem [CITE], this means that Life is undecidable. Given two configurations, it is impossible to algorithmically determine whether one will follow the other.

Patterns found within Life include still lifes like the *block* which are fixed-point solutions to the transition function as well as periodic oscillators like the *beacon* which has period 2. There are also periodic patterns that move across the lattice such as the *glider* pattern. It is possible to discover new stable patterns by repeatedly running specific rules on random initial patterns of a pre-determined density (called soups) and classifying the objects remaining after transient reactions have dissipated. Large-scale experiments of this nature are called "soup searches"[CITE].

A CA is considered "Life-like" if it exists on a 2D lattice, has binary state, uses the Moore neighbourhood function. Life-like cellular automata exist in two varieties: inner-totalistic and outer-totalistic.

Definition (Inner-totalistic)**.** *A Life-like CA is inner-totalistic if the output of the transition function depends only on the number of living cells in a cell's neighbourhood (including the cell*

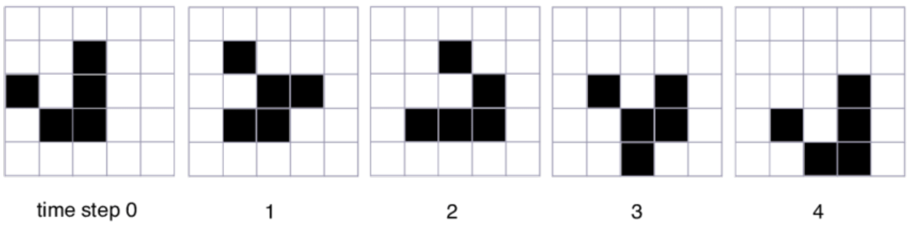


Figure 2.2: The glider pattern in the Game of Life [2]

itself).

$$\sigma_i(t+1) = \sigma_j(t+1) \iff \sum_{c_p \in \mathcal{N}(c_i)} \sigma_p(t) = \sum_{c_q \in \mathcal{N}(c_j)} \sigma_q(t)$$

Definition (Outer-totalistic)**.** *A Life-like CA is outer-totalistic if the output of the transition function depends on both the number of living cells in a cell's neighbourhood and the state of the cell itself.*

$$\sigma_i(t+1) = \sigma_j(t+1) \iff \sum_{c_p \in \mathcal{N}(c_i)} \sigma_p(t) = \sum_{c_q \in \mathcal{N}(c_j)} \sigma_q(t) \quad \text{and} \quad \sigma_i(t) = \sigma_j(t)$$

As an example of the subtle difference here, consider the configurations shown in Figure 2.3. An inner-totalistic CA would yield identical configurations in the next time step since both input configurations have 3 active cells in the neighbourhood set. However, an outer-totalistic CA would treat both configurations separately as one has an live centre cell and the other has a dead centre cell. This discrepancy corresponds to a great difference in the size of search spaces. There are $2^{10} = 1024$ inner-totalistic CA but $2^{18} = 262144$ outer-totalistic CA. A B/S rulestring represents the transition function of an outer-totalistic CA in a form called birth-survival notation.

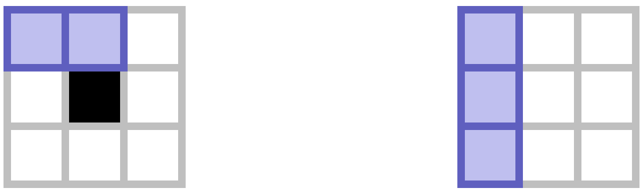


Figure 2.3: Two possible configurations of a Life-like CA[CITE]

Definition (Birth-survival notation)**.** *Let N_b and N_s be sets of integers. We say an outer-totalistic CA has rulestring* BN_b/SN_s *if it has transition function:*

$$\phi(\sigma_i(t), n) = \begin{cases} 1 & \sigma_i(t) = 0 \textit{ and } n \in N_b \textit{ (Birth)} \\ 1 & \sigma_i(t) = 1 \textit{ and } n \in N_s \textit{ (Survival)} \\ 0 & \textit{otherwise} \end{cases}$$

Using this notation, we can represent the Game of Life as B3/S23. In this thesis, when we refer to Life-like CA, we implicitly assume the outer-totalistic variety.

2.1.3 Wolfram's Classification

The choices of lattice geometry, neighbourhood function, state variable, and transition rule define the behaviour of a CA. Fixing the former three factors, Wolfram [8] 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 $(t_7 t_6 t_5 t_4 t_3 t_2 t_1 t_0)$ 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.

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 [3] and is Turing complete [9]. Figure 2.4 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.

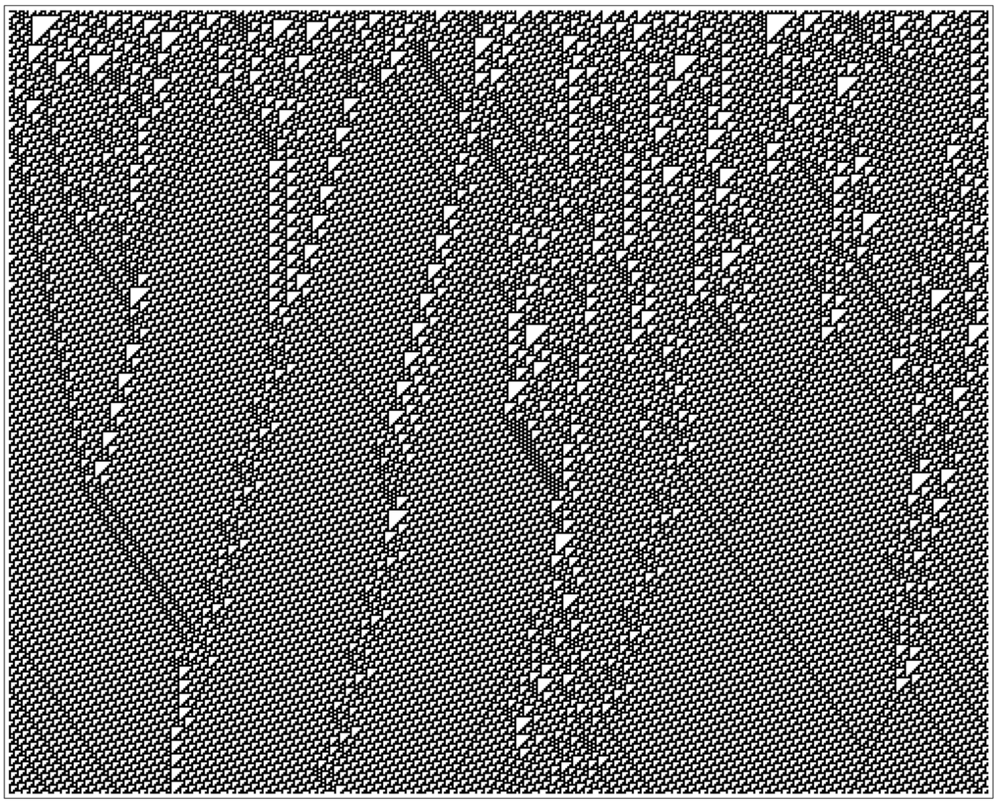


Figure 2.4: Rule 110 progression with random initialisation [3]

2.2 Evolutionary Algorithms

Evolutionary algorithms (EAs) are a family of heuristic-based search algorithms for black-box optimisation problems. They are inspired by biological evolution. A population of candidate solutions is initialized and modified through repeated selection, mutation, and reproduction. EAs are valued for their broad applicability as they require no information about the constraints or derivative of the objective function. In fact, an explicit representation of the objective function is not even neccessary to run an EA as long as candidates can be compared to each other. Selection pressure can then be introduced in the form of tournament-based elimination.

Typically, an EA acts on a population of "chromosomes" which are indirect encodings of candidate solutions. The structure of the chromosome is called the "genotype" and the structure of the corresponding solution is called the "phenotype". In this thesis, the genotype will usually be a set of parameters that characterise the transition function for a particular class of CA. The corresponding phenotype is the cellular automaton with that transition function.

Chapter 3

Related Works

3.1 Learning cellular automaton dynamics

Learning Cellular Automaton Dynamics with Neural Networks (Wulff and Hertz, 1992) [10] 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 [11]. 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 cases. 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) [4] 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 patterns 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.

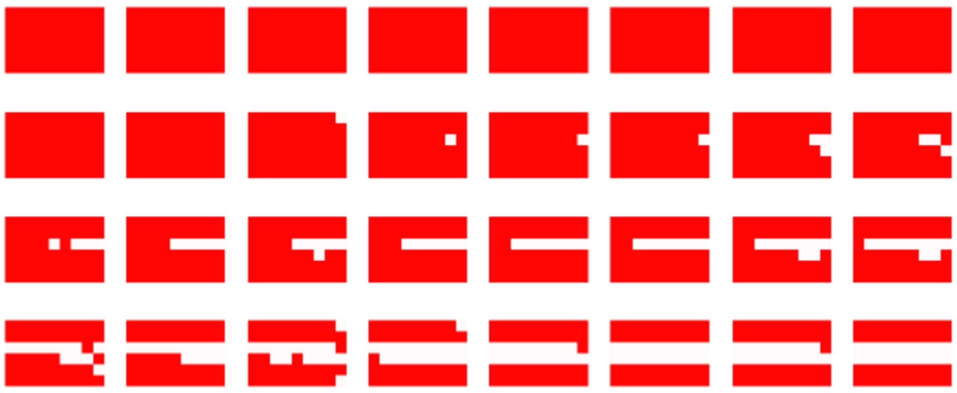
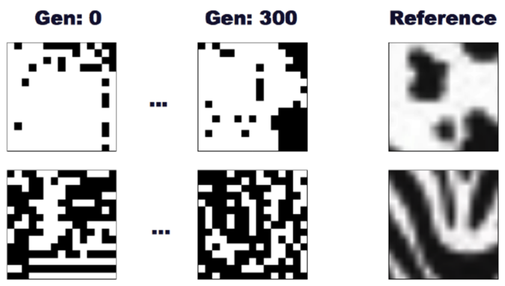
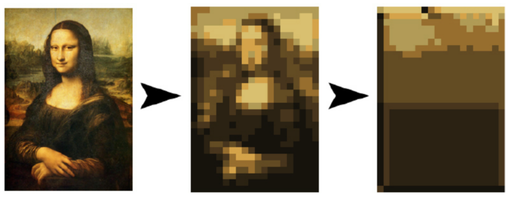


Figure 3.1: Successful reproduction of the Austrian Flag over 90 generations [4]



(a) Attempt at learning cow and zebra skin patterns over 300 generations [4]



(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 [4]

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) [12] 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 [13]. 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 separate species once they cross a threshold 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 proportion 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.

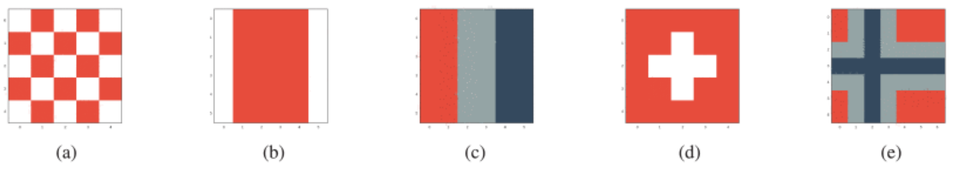


Figure 3.3: Five goal patterns for CA-NEAT

(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 [14] 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.

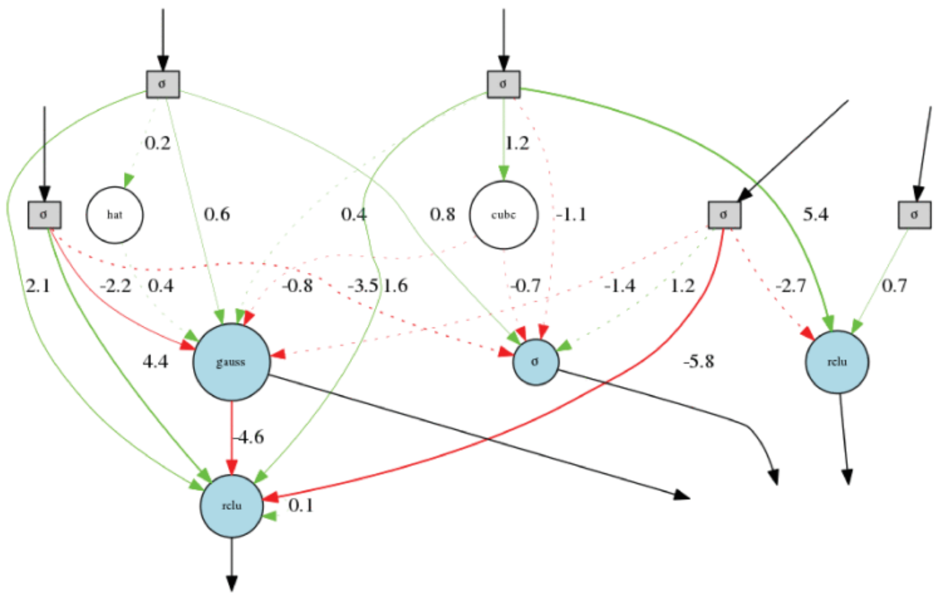


Figure 3.4: fixed-point CPPN solution found for *Tricolor* morphogenesis

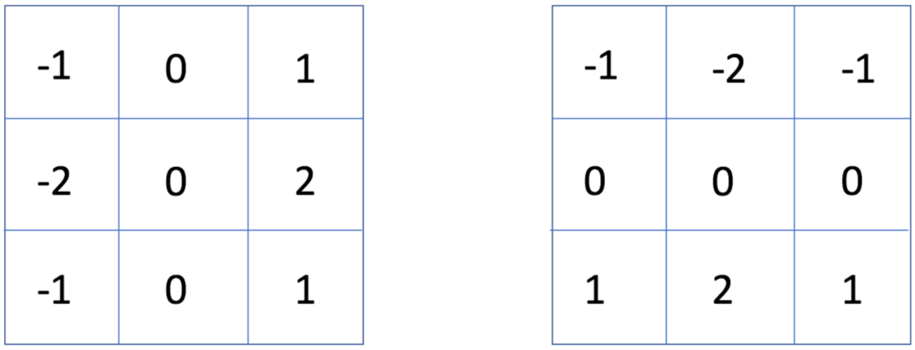
Dotted lines indicate disabled connections. Note the two vestigial hidden layer nodes

3.3 Using convolutional neural networks

Growing Neural Cellular Automata (Mordvintsev et al., 2020) [15] 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 cells. 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 (see Figure 3.5 are used to 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 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.



-1	0	1
-2	0	2
-1	0	1

-1	-2	-1
0	0	0
1	2	1

Figure 3.5: Sobel filters in the x-direction (left) and y-direction (right) [5]

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 is iterated using the network. The individual with the highest loss in the batch is reset to the seed state to prevent catastrophic forgetting [16]. 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 achieve persistance 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.

Overall, this paper was a very successful breakthrough in using artificial neural networks for morphogenesis. It opened up many possible areas of exploration including self-classifying CAs [17], adversarial attacks on morphogenetic CAs [18], and learning graph CA [6].

3.4 Using graph neural networks

Learning Graph Cellular Automata (Grattarola et al., 2021) [6] 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 perceptrons (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 a rule which flips the binary state of a cell depending on the number of living neighbours. This is analogous to Conway's Game of Life except on a Voronoi tessellation (see Figure 3.6) instead of a regular lattice. This is fairly easy since there is an explicit, known target rule and the graph topology is static.

2. Boids : Learn Reynold's algorithm [19] 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 has 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.

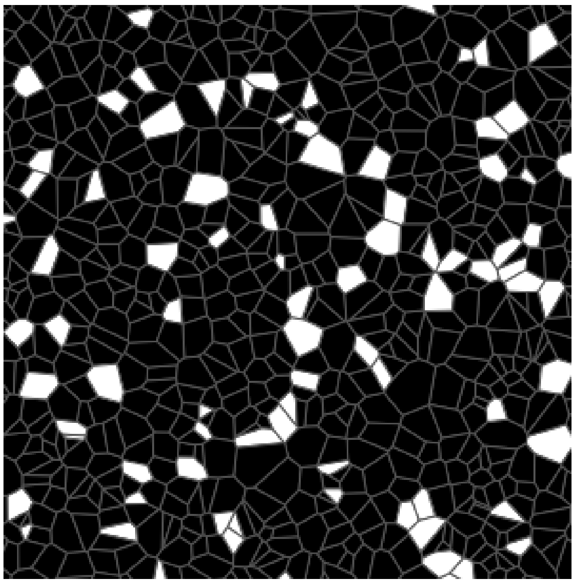


Figure 3.6: A GCNA learning on a Voronoi tessellation [6]

This work was successful at discovering rules that can converge and persist complex target graphs including the Stanford Bunny [20] 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 and, naturally, we can consider fixed-point solutions to be a subset of periodic solutions with a period of 1. However, for the goal of producing stable shapes through morphogenesis, we are interested in discovering ways to separate these two types of solutions. We can speculate that the model converges to periodic solutions at times because they are "close" to fixed-point solutions in the rule space.

Chapter 4

Life-like Cellular Automata

4.1 Genetic Algorithm

4.2 Species Analysis

Chapter 5

Maze Generation

5.1 Region Merging Algorithm

5.2 Fitness Evaluation

5.3 Selection and Quality-Diversity

Chapter 6

Gray-Scott Models

6.1 Evolutionary Strategies

6.2 Particle Swarm Optimization

Chapter 7

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 develop 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 [15] to the context of graph CA introduced by Grattarola et al. [6]), 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.

7.1 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 [10], this was a relatively stagnant area of research until the breakthrough work by Mordvintsev et al in 2020 [15].

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 [6] 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 [15].

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 [21]. 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 separate 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 to improving and polishing the final report.

Chapter 8

Conclusions

Chapter 9

Evaluation

9.1 Ethics

As a relatively abstract mathematical project, there are no major ethical factors to consider here. However, there are still some legal and societal issues that are worth discussing, especially given that technologies falling in the broad field of self-organising systems, distributed systems, and automated systems can be misused.

The only area of legal concern is the infringement of copyright law when selecting training data. To train graph cellular automata, we many use point cloud datasets. These tend to be of physical objects or terrain maps but can, in theory, be derived from any image. When choosing point clouds to train on, we will ensure that they do not fall under copyright restrictions that make them unsuitable for academic use.

The only area of societal or professional concern is the general application of cellular automata in distributed technology like swarm robotics. Such systems can have military applications. It could be argued that this research could pave the way for highly distributed swarms of drones or ground robots that are robust to partial damage or perturbation. However, this is unlikely to be a true concern since we only discuss theoretical concepts in this paper with little discussion about real-life applications. Furthermore, this research is only in its preliminary stages and the academic benefits of researching self-organising cellular automata far outweigh the negligible potential contribution that this research could have to improving malicious swarm robotics systems.

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] 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.

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

[4] 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.

[5] Sean Sodha. An implementation of sobel edge detection. URL https://www.projectrhea.org/rhea/index.php/An_Implementation_of_Sobel_Edge_Detection.

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

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

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

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

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

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

[12] 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.

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

[14] 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.

[15] 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.

[16] Michael McCloskey and Neal J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. volume 24 of *Psychology of Learning and Motivation*, pages 109–165. Academic Press, 1989. doi: https://doi.org/10.1016/S0079-7421(08)60536-8. URL https://www.sciencedirect.com/science/article/pii/S0079742108605368.

[17] 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.

[18] 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.

[19] 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.

[20] Greg Turk. The stanford bunny. URL `https://faculty.cc.gatech.edu/~turk/bunny/bunny.html`.

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