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 17, 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 for 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 optimisation problems from the imitation of particular CA behaviour to generation of desirable long-term states.

1.2 Objectives

We aim to develop a system to discover cellular automata that are highly likely to exhibit a desired behaviour.

1.3 Contributions

The key contributions of this project are as follows:

1. **Evolutionary Algorithm Toolkit**
   A versatile toolkit that implements multiple evolutionary algorithms to train and optimise different classes of CA. We use this to successfully predict the update rule of binary outer-totalistic CA from observations of the CA running on random initial conditions. We show that this can be extended to continous automata by predicting the parameters of diffusion-reaction equations from simulations of chemical reactions.

2. **Cellular Automaton Simulator**
   A system that can efficiently simulate discrete and continuous cellular automata. This allows a broad range of fitness functions to be implemented in the EA toolkit. This can also render snapshots of the CA directly during simulation which allows animations to be efficiently generated afterwards.

3. **Procedural Maze Generator**
   A CA-based maze generation program that uses the EA toolkit to produce difficult mazes with characteristics optimised to user preference.

1.4 Technical Challenges

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.

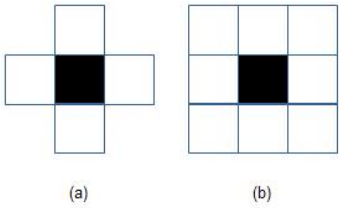

Figure 2.1: (a) von Neumann Neighbourhood and (b) Moore neighbourhood on a 2D square lattice [1]

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 [4]. 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*

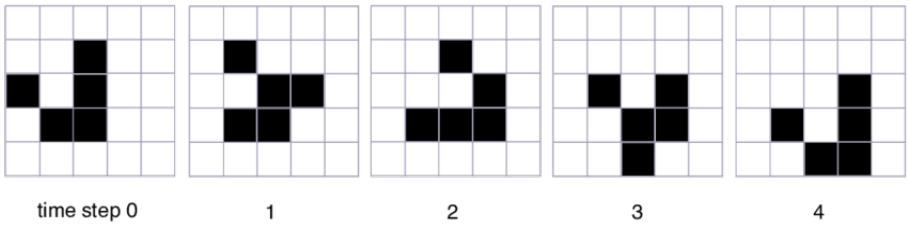

Figure 2.2: The glider pattern in the Game of Life [2]

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:*

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

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 [5] 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 [6]. 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.

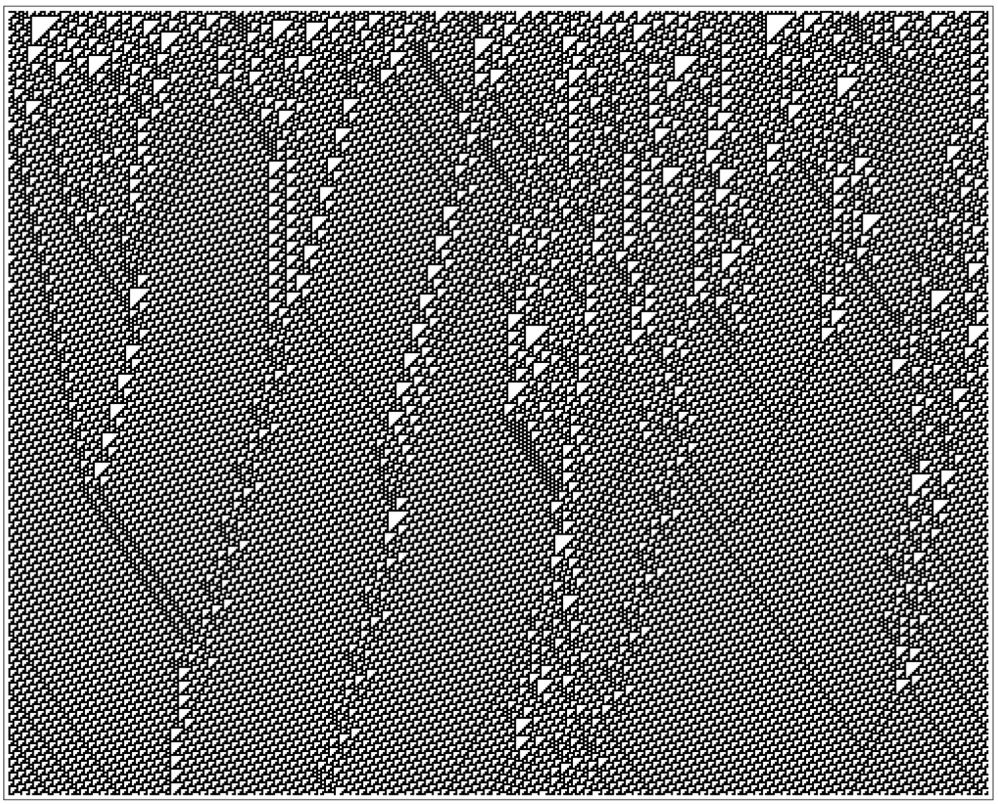

Figure 2.4: Rule 110 progression with random initialisation [3]

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 recombination. We define an EA formally in Algorithm 1.

Algorithm 1 Schematic Evolutionary Algorithm

Require: S - the set of possible chromosome values
Ensure: $s^* \in S$
  $t \leftarrow 0$
  $M_0 \leftarrow \mu$ random individuals from S
  while stopping condition is false **do**
    EVALUATE(M_t)
    $P_t \leftarrow$ SELECTPARENTS(M_t) ▷ Parents
    $\Lambda_t \leftarrow$ RECOMBINE(P_t) ▷ Children
    $Pmod_t \leftarrow$ MUTATE(P_t)
    $\Lambda mod_t \leftarrow$ MUTATE(Λ_t)
    $M_t + 1 \leftarrow$ SELECTPOPULATION($Pmod_t$, Λmod_t)
    $t \leftarrow t + 1$
  end while
  $s^* \leftarrow$ FINDBESTCANDIDATE(P_t)

The initial selection phase (SELECTPARENTS()) uses a objective function, also known as a fitness function, to compare and select the top candidates. Recombination produces a set of children that have similar properties to some subset of the parents. This exploits the cumulative progress of the evolutionary process embedded in the parent candidates. Mutation explores new areas of the search space by perturbing properties of the parents and children. The latter selection phase (SELECTPOPULATION()) produces a new population from the modified parents and children. Population-wide selection criteria can be enforced in this phase. For example, certain parents can be eliminated if they have survived for too many generations or, symmetrically, children can be granted immunity for a particular number of generations.

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 Discrete CA with Genetic Algorithms

Learning Algorithm for Modeling Complex Spatial Dynamics (Meyer, Richards, and Packard, 1989) [7] is a seminal work in this area. It creates a dynamical model from observations of discrete values evolving on a lattice over time. The model is represented as a binary probabilistic cellular automaton (PCA) whose neighbourhood set is learned by a genetic algorithm. The motivation for this work is to establish the form of CA which would most accurately describe patterns in the physical world directly from experimental data of physical interactions. In particular, Richards et al. [8] builds on these techniques to find PCA rules that accurately predict the evolving patterns generated by the dendritic solidification of NH_4BR.

Note that this goal is different from learning the entire transition function of a cellular automaton. This work merely aims to establish *which* parameters in a local vicinity of the current state are most relevant to predicting the future state, not *how* those parameters are combined and transformed to produce the future state.

The aim is to discover the most appropriate local neighbourhood set within a vicinity from the central cell that extends two steps in both space and time. This is the intersection of the Moore neighbourhood in time step $t-1$ and the von Neumann neighbourhood of range 2 in time step $t-2$. This comes to a 20 cell neighbourhood in space and time as visualised in Figure 3.1.

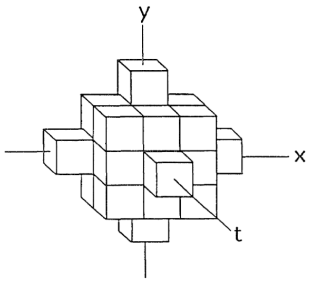

Figure 3.1: 20-cell, two step neighbourhood in space and time

The full 20-cell neighbourhood is called the master template and each chromosome encodes some subtemplate $s_1, ..., s_m$. The fitness function used is

$$F = I - \frac{2^m}{N}$$

$$\text{where } I = \sum P(s, s_1, , ..., s_m) \log_2 \frac{P(s, s_1, ..., s_m)}{P(s)P(s_1, ..., s_m)}$$

Here, I is the mutual information of the subtemplate and represents the amount of information (measured in Shannon bits) about the value of the central cell that can be obtained from the states of the cells in the subtemplate. It is calculated by summing across all 2^m configurations of

the subtemplate in the data and across both values of $s \in \{0, 1\}$. The second term in the fitness function ensures that subtemplates of varying sizes are treated appropriately by proportionately penalising large subtemplates that, by nature, will contain more information. In our case, $N = 20$

The genetic algorithm initialises the population randomly. At each time step, a linear ranking of candidiates is performed and truncation selection is performed to pick the fittest subset. Crossover is applied between pairs of randomly selected candidates where the crossover point is an arbitrary cut in space-time on the master template. Point mutation is applied by either adding or removing a single cell from each candidate. This process is iterated to converge towards an optimum.

This method was successful at learning test data generated from a 1 time step Moore neighbourhood and data generated from 1 time step von Neumann neighbourhood with range 2. The algorithm successfully reproduces patterns generated by the first dataset precisely. However, unlike the first goal, the second goal neighbourhood is not subsumed by the master template. This means the algorithm can only hope to learn a similar template, not the exact objective. Despite this, the algorithm was able to find a neighbourhood set that produced correct behaviour 96% of the time.

As the first notable exploration of learning CA properties with genetic algorithms, this paper devised a novel method for learning neighbourhood sets on binary PCA. It demonstrates the ability of this algorithm to predict neighbourhoods interior to the allocated search space as well as close approximations for objectives outside the search space.

This work raised many questions for future research. The most pertinent is whether it is possible to link learned rules to existing and future theoretical models. This work also only explored binary state CA but this excludes many continous variables relevant to data obtained from physical reactions such as the temperature field. An exploration of similar techniques on continous-state CA could be explored to closer approximate the partial differential equations that underlie the processes being explored.

Finally, this paper focused only on identifying the neighbourhood set of the CA that would closely approximate the interactions being studied. For the purposes of this thesis, we are interested in going beyond this and approximating the full transition function. In some cases we will fix the neighbourhood function used to reduce our search space under the assumption that further research could use techniques outlined in this paper to test whether better sub-neighbourhoods exist.

Chapter 4

Design

In this chapter we outline the workings of the evolutionary algorithm toolkit.

4.1 Evolutionary Algorithm Toolkit

4.2 Cellular Automata Simulator

Chapter 5

Method

5.1 Life-like CA

5.1.1 Genetic Algorithm

5.1.2 Species Analysis

5.2 Reaction-diffusion CA

5.2.1 Evolutionary Strategies

5.2.2 Particle Swarm Optimisation

Chapter 6

Application

6.1 Maze Generation

6.1.1 Region Merging Algorithm

6.1.2 Fitness Evaluation

6.1.3 Selection and Quality-Diversity

Chapter 7

Evaluation

Chapter 8

Conclusions

Chapter 9

Ethical Considerations

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

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

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

[7] Thomas P Meyer, Fred C Richards, and Norman H Packard. Learning algorithm for modeling complex spatial dynamics. *Physical review letters*, 63(16):1735, 1989.

[8] Fred C Richards, Thomas P Meyer, and Norman H Packard. Extracting cellular automaton rules directly from experimental data. *Physica D: Nonlinear Phenomena*, 45(1-3):189–202, 1990.