Program Cells in Cellular Automata

Jack Schenkman

April 2026

1 Introduction

Cellular automata represent a simple model of local interaction capable of producing incredibly complex patterns in space and time. Since von Neumann the properties of cellular automatas ranging from neighborhoods to rule classification to conditions for self-replication have been thoroughly studied. Providing a survey of this field is beyond the scope of this work. This research explores the use of cellular automata as computing devices, meaning lattices of cells that can collectively perform a desired function. One could argue that this definition means nothing, considering that practically everything a CA does can be described as a function. Unlike efforts to create replicators or specific geometric patterns, this research is focused on obtaining a CA that obeys specific input-output relations. Most of the previous work attempting to leverage cellular automata to perform specific computations has focused on using a genetic algorithm to evolve the rules of the CA [3][2]. Differentiable logic cellular automata utilize gradient descent to optimize the rules enabling the discovery of CAs that perform complex computations [1]. In this research the CA rules are fixed and the implementation of different functions is achieved by changing the state of specific cells.

2 Program Cells

This work explores whether the desired computation can be achieved by clamping certain cells within the lattice, referred to as program cells, to particular values without modifying the CA rules. Traditionally, the CA rules define the computation being performed, and the state of each cell defines intermediate variables or the final result of the computation. In this research, the boundary between the representation of the data and the computation instructions is removed. Such boundaries are in many ways semantic. Clamping the state of cell A is the same as modifying the truth table of its neighboring cell B to be limited to a subset of the full 128-row truth table in which cell A has the specific value to which it was clamped.

3 Program Cell Experiment

3.1 Lattice

The goal of the experiment is to clamp program cells in a configuration that caused the lattice to implement a random 4-bit Boolean function. More specifically, when a specific set of 4 bits are presented to the lattice, the lattice should output the correct value according to the randomly chosen Boolean function. The experiment uses an 8 x 8 x 8 lattice.

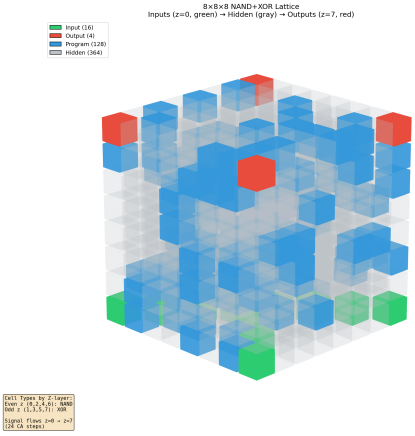
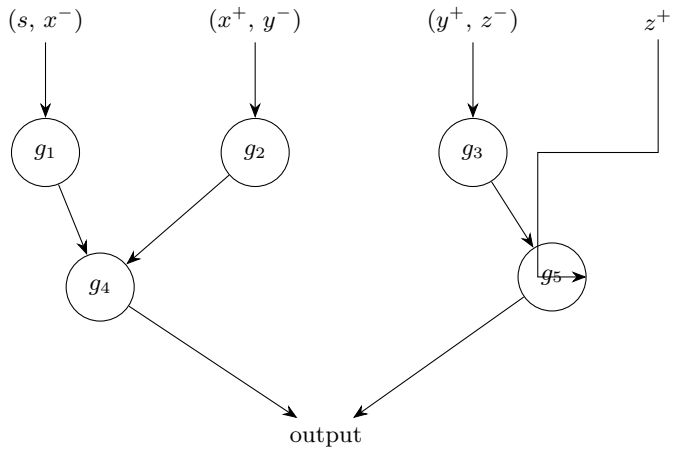


Figure 1: 8 x 8 x 8 lattice with cells labeled as input, program, hidden, and output.

Each cell implements a 7-bit Boolean function (6 inputs are from the neighbors and one input is from itself). The cell is defined by a $2^7 = 128$ row truth table. A toroidal topology is used such that cells on the edge receive wraparound inputs. The bottom face of the lattice has 4 cells chosen as the inputs. For a given input, these cells are clamped. The opposite face has 4 cells chosen as outputs. The 4 output cells are averaged together to determine the output. Of the remaining cells within the lattice, 128 are chosen as program cells, meaning their values are clamped to cause the lattice to compute the desired 4-bit Boolean function. The Boolean function implemented by a given cell is represented as a tree shown below.



In this diagram, x refers to neighbors along the x-axis, y refers to neighbors along the y-axis, z refers to neighbors along the z-axis, and s refers to the state of the cell. The functions g_i are identical for all i in the experiments that are performed for a given cell. For cells in even layers g_i is NAND and for cells in odd layers g_i is XOR. The choice of NAND and XOR was motivated by the universality of NAND and the linear mixing from the XOR. A first-principles approach was not used to determine the truth tables.

3.2 Genetic Algorithm

A genetic algorithm is used to find the configuration of program cells that will cause the CA to best approximate the randomly chosen 4-input Boolean function from a randomly generated set of 100 such functions. Finding the configuration of program cells for a given 4-input Boolean function is its own search process. The genetic algorithm search space consists of the states of the 128 program cells. A population of 6000 lattices was used with tournament selection size of 7, mutation rate of 0.03, and crossover probability of 0.8. Other numbers of program cells were tried and tying some of the cells together such as using 64 different program bits but assigning two cells to each bit was also tried. These efforts were inferior to using 128 program cells.

3.3 Results

For all 100 of the randomly generated 4-input Boolean functions, the genetic algorithm found a configuration of program cells that yielded 100 percent accuracy on outputting the correct value in response to each possible input for the function.

4 Conclusion

This research demonstrates the ability of a cellular automata to perform various functions specified by their I/O behavior simply by clamping cells within the lattice. Questions remain regarding the optimal location of the program cells, the neighborhood size, and even the choice of a CA based on local connectivity rather than a more general Kauffman-style N-K network. There is room for optimizing or more intelligently choosing the truth tables such that the genetic algorithm has an easier time finding good program cell configurations.

References

[1] Pietro Miotti, Eyvind Niklasson, Ettore Randazzo, and Alexander Mordvintsev. Differentiable logic cellular automata: From game of life to pattern generation. In *Artificial Life Conference Proceedings 37*, volume 2025, page 54. MIT Press One Rogers Street, Cambridge, MA 02142-1209, USA journals-info . . . , 2025.

[2] Melanie Mitchell, James P Crutchfield, and Peter T Hraber. Evolving cellular automata to perform computations: Mechanisms and impediments. *Physica D: Nonlinear Phenomena*, 75(1-3):361–391, 1994.

[3] Melanie Mitchell, Peter Hraber, and James P Crutchfield. Revisiting the edge of chaos: Evolving cellular automata to perform computations. *arXiv preprint adap-org/9303003*, 1993.