Genetic Algorithm for Evolving a CA to Solve the Majority Problem

Adam Callahan

August 6, 2009

1 Introduction and Background

1.1 Introduction

Continuing exploration of cellular automata, computational methods of searching large spaces, and parallelizable code, we explore genetic algorithms and reproduce some of the work of Melanie Mitchell, Peter Hraber, and James Crutchfield. This work uses a genetic algorithm to evolve a cellular automata to solve the "majority problem".

1.2 Background

A cellular automata (CA) consists of a discretized space (definining cells) and a set of rules that determine the value of each cell at the next time step. This set of rules depends on the values of the cells in the local neighborhood and sometimes the value of the cell itself for a certain number of time steps. Usually, the number of time steps is one and the neighborhood is small. For a one dimensional CA, this may be written as:

$$c_i(t) = F(c_{i-r}(t-1), c_i r + 1(t-1), \ldots, c_{i+r-1}(t-1), c_{i+r}(t-1))$$

where $c_i(t)$ is the value of the cell c indexed by i at time t and r is the radius of the neighborhood.

The "majority problem" can be stated as:

Given a vector of length N that consists of 0s and 1s with density of 1s ρ, find a CA with $r \ll N$ such that the CA progresses to a fixed point that consists of all 1s if $\rho > 1/2$ and all 0s if $\rho < 1/2$.

The "majority problem" is trivial for a Von Neumann architecture computer, but this is not the case with a CA. A CA has a parallel architecture that does not necessarily transmit memory or instructions over arbitrarily large distances. The problem is not trivial to solve by hand. A radius of 1 can not come close to solving this problem. Mitchell et al use a CA of radius 3. The problem here is that the possible solution space is huge and it is not easy to understand how the rules will evolve the initial conditions. Since each cell can have 2 values and 7 cells values are taken into account, the possible solution space contains $2^{2^7} = 2^{128}$ possible solutions. To build the rule for the CA, we can use the method of Stephen Wolfram, expanding it to allow a larger neighborhood. The method is for "elementary CA", that is, a one dimensional CA with neighborhood radius one and whose cells take on only two values. The method is as follows. Treat the possible neighborhoods values as if they were binary numbers and list them in order. For each possible neighborhood, list the updated value of the cell. Treat this list as a binary number. Thus, the elementary CA with rule 90 is:

binary neighborhood	updated cell value
111	0
110	1
101	0
100	1
011	1
010	0
001	1
000	0

For a CA with a neighborhood of radius 3, the strings of possible neighborhood values will be of length 7. Accordingly, there will be $2^7 = 128$ possible neighborhoods. The update rule of the CA can be coded as a binary number with 128 digits.

This solves the problem of coding the possible solutions to the problem in a suitable manner for a genetic algorithm. A genetic algorithm is a method of searching a possible solution space. The method consists of:

1.) Code the possible solutions in some numerical manner such that any change in the code will result in a change in the solution and all solutions can be attained by this encoding.

2.) Generate a population of solutions.

3.) Evaluate the population via some "fitness" function.

4.) Modify the population (usually by combining successful chromosomes (mimicking sexual reproduction) and mutating them).

5.) Repeat steps 3 and 4 for a preset number of "generations" or until some goal is met.

The reproduction occurs by choosing two "chromosomes", choosing a random location in the "chromosome" and breaking it at that point. The first portion of one chromosome is followed by the second portion of the other chromosome. The opposite occurs for the other "child". Mutation occurs by selecting a small number of "genes" and flipping their values.

parent 1	parent 2		child 1	child 2
1	0		1	0
1	1		1	1
0	0		0	0
⋮	⋮	$\Longrightarrow$	⋮	⋮
1	0		0	1
0	1		1	0
1	0		0	1

2 Methodology and Results

We use different methods and code in an attempt to evolve the best CA rule. The method described in Mitchell's "Introduction to Genetic Algorithms" is different than that described in her paper. We attempt to emulate sections of each.

2.1 Method I

The first method constructed vectors for the initial condition of the CA in the following manner. Ten vectors were created at each discrete probability value x, where $x \in \{0.05, 0.15, 0.25...0.85, 0.95\}$. Thus 100 initial conditions were created at the beginning of the program. An initial population of chromosomes was generated with each cell of the chromosome having a probability of 0.5 of being assigned a value of 1, otherwise being assigned a value of 0. Thus, the chromosomes were taken from a distribution with the density of 1s clustering around 0.5. Each chromosome was run through each initial condition. If the density of 1s for the initial vector was above 0.5 and the final row of the evolved CA consisted of all 1s, the chromosome's fitness was increased by one. If the density of 1s was below 0.5 for the initial vector and the final row of the evolved CA consisted of all 0s, the chromosome's fitness was increased by one. Otherwise, the chromosome's fitness remained unaltered. No partial credit was awarded. The CA was evolved for a set number of steps equal to twice the initial vector's length. For most runs, including the run that produced the results below, the initial vector's length was 49. The GA kept the top 20 performers, and then produce "children" by uniformly randomly selecting two "parent" chromosomes from the top performers. These chromosomes were then "mutated" by switching from 0 to 2 "genes".

Interestingly, the results of this first attempt demonstrate a fairly good strategy of classifying the initial vector's density of 1s. It appears that, if the local density of ones is high enough, the rule expands the cluster at highest speed on the left side and expands the cluster at a slightly slower speed on the right side. This is not very dissimilar from the GKL rule. The GKL rule is one of the best rules known for solving this problem.

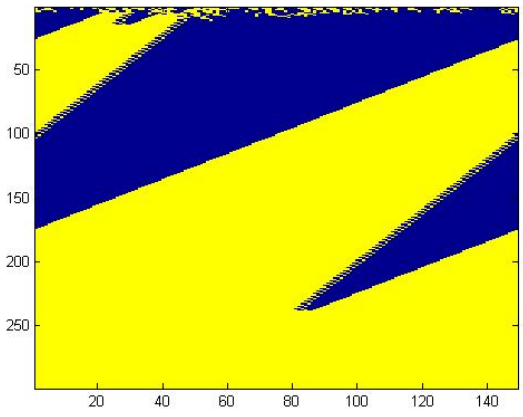

Figure 1: Correctly classified initial condition

However, this is not the best strategy, and the rule fails for a significant number of initial conditions. It's fitness over all initial conditions was 92. However, when the initial vector has close to density 0.5, it's performance drops dramatically.

2.2 Method II

The only difference for the second method was to keep only those chromosomes that had a fitness greater than 1. The difference appears to be significant, however, as the average performance of the CAs dropped. Illustrated below is a typical resulting rule.

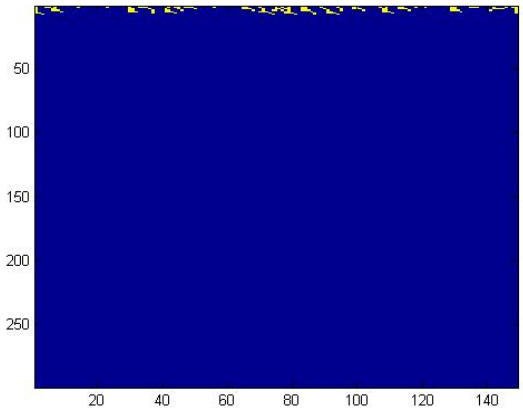

Figure 2: Incorrectly classified initial condition

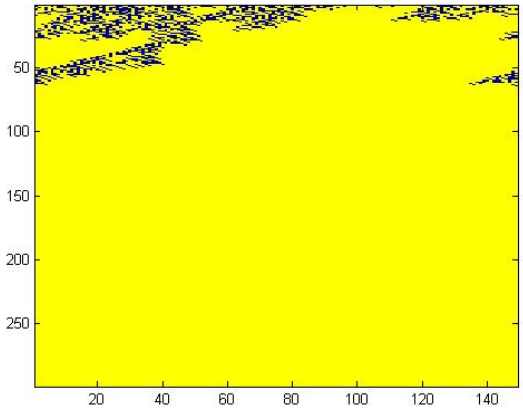

Figure 3: Correctly classified initial condition

As can be seen, the rule almost invariably evolves to the fixed point of all ones. This results in a fitness of close to 0.5 for initial conditions that have a density of ones close to 0.5. As discussed earlier, most initial conditions generated as we did are close to this density.

2.3 Method III

Several modifications were made for the third coding. First, the number of initial conditions was increased to 300 from 100. The length of the CA row was increased to 149 from 49. Also, the number of initial conditions with density $\rho > 0.5$ was exactly equal to the number of initial conditions with density $\rho < 0.5$. This was implemented to ensure that there would be no initial bias in the rules. The GA now included the top 50 chromosomes whose fitness was above 20 (evaluated on a scale of 0 - 300). The other chromosomes were generated by mating two randomly selected chromosomes from the entire population, not just the top performers. This was introduced to attempt to prevent the GA from settling on a solution too early. The

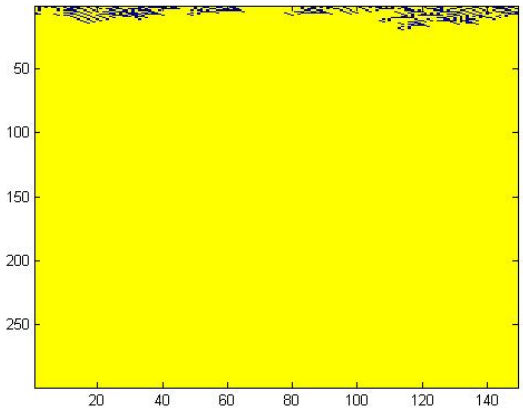

Figure 4: Incorrectly classified initial condition

mutation rate was also increased to flipping from 0 to 4 genes. The GA appeared to settle on a few solutions relatively soon, despite these precautions. In fact, the top 50 chromosomes had the same fitness after only 3 generations on the majority of the runs. The GA was run for only 30 generations due to time constraints. Future work will include increasing this to 100 generations. A slight variation on the initial conditions was also attempted, though the results were no better than the results of the unaltered version of this method. The difference was to change the density of the initial conditions with the generation to exclude density values distant from 0.5. Specifically, the density of the initial conditions started with a uniform distribution and exponentially got closer to 0.5 as the generation index approached 30.

The results of this third method were disappointing. It appears that almost all initial conditions evolve to a quiescent state of all 0s.

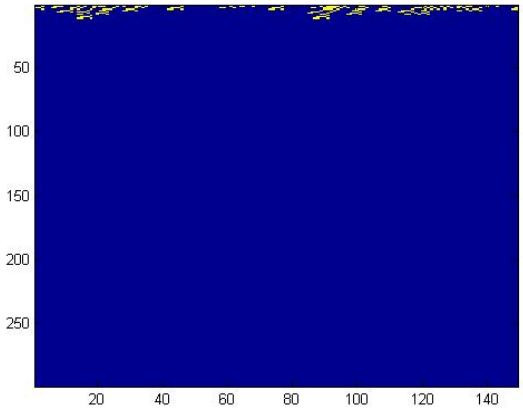

Figure 5: Correctly classified initial condition

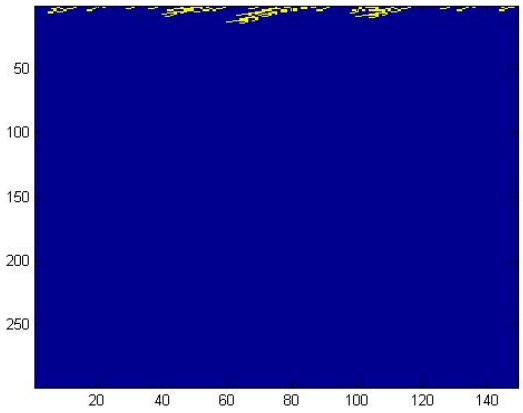

Figure 6: Incorrectly classified initial condition

3 Future Work

Future work will include exploring different lengths of the initial condition, changing the density of the initial conditions as the GA evolves, changing the density of the initial chromosomes to exclude those chromosomes with density far from 0.5, increasing the number of generations the GA evolves for, and parallelizing the code using open mp. The results of this work will be documented and posted on a blog at wordpress.

4 References

[1] M. Mitchell, P. Hraber, and J. Crutchfield. Revisiting the Edge of Chaos: Evolving Cellular Automata to Perform Computations. Physica D, 75:361-391, 1994.

[2] M. Mitchell. An Introduction to Genetic Algorithms. MIT Press. Cambridge, Massachusetts, 1996.