

# Searching for a Synchronizing 2-D Cellular Automata

Daniel Kunkle

April 9, 2002

## 1 Introduction

This work is an extension of research done by the Evolving Cellular Automata (EvCA) group at the Santa Fe Institute. Specifically, it is an extension of their application of genetic algorithms (GAs) to design cellular automata (CAs) that synchronize globally while using only local information [1]. In the work of Das et. al, 1-D CA transition tables for a neighborhood of size seven (radius of 3) are searched with a GA. The fitness of the genome representing the transition table is based on its ability to synchronize the CA starting with a random initial state. Synchronization simply means that all of the cells change from state 1 to state 0 in unison.

In this research, a 2-D CA is sought that can perform the same synchronization task. Similar to the work done in 1-D, the 2-D grid is toroidal so that no unusual boundary conditions exist. This task is made more difficult by two main factors:

1. The interactions of the cells in 2-D is more complex than that of cells in 1-D. The simple “particle” explanation explored in 1-D CA synchronization does not have a simple counterpart in 2-D CA synchronization tasks.
2. The neighborhood size is increased from seven in the 1-D CA to nine in the 2-D CA. A neighborhood of size nine includes the cell itself, all cells that directly share faces with that cell, along with all cells diagonal from that cell. This configuration was chosen because it is the traditional configuration for 2-D CAs, though it is possible that other configuration can perform the synchronization task. With an increase of neighborhood size from seven to nine the length of the genome increases by a factor of 4 (*length of genome* =  $2^{\text{neighborhood size}}$ ). With a genome of bits, this results in an increase of the size of the search space from  $2^{128}$  to  $2^{512}$ .

Despite these added complexities the GA was able to evolve CA transition tables to synchronize given initial random states.

## 2 Goals

Primarily, this research is conducted to identify CA transition tables that synchronize given arbitrary CAs of different sizes and initial state. More specifically, parameters and methods of GAs and CAs are examined as they relate to solving this puzzle. To these ends tools were constructed to allow experimentation with different parameters and methods as well as visualizations of 2-D CAs to allow the experimenter to identify at a high level how the CA synchronizes.

## 3 A First Attempt By Hand

A simplistic approach to developing a CA transition table leading to synchronization is to construct it manually. The most obvious of such rules is: If a cell has more neighbors in state 0 at time  $t$  take on state 1 at time  $t + 1$ , otherwise take on state 0 at time  $t + 1$ . So, the cell assumes that all of its neighbors are trying to synchronize, oscillating between state 0 and 1 each time step. If more neighbors are at state 0 at

$t$ , they should be in state 1 at  $t + 1$ , and so should the cell. If the genome is constructed such that the first half represents the condition where more neighbors are in state 0 and the second half where more neighbors are in state 1 the genome  $G = g_1, g_2, \dots, g_{512}$  is simply  $g_1 \dots g_{256} = 1, g_{257} \dots g_{512} = 0$ .

This simplistic strategy does poorly, no initial states were found that were synchronized with this transition table. When viewing a time sequence of the execution of the CA it became apparent that the system would very quickly find non-synchronized cyclic attractors.

## 4 Attractors

A CA is an instance of a dynamical system, a system that evolves through time according to given evolutionary rules. An attractor in a dynamical system is a state, or set of states, that the system settles into. Once the system has settled into an attractor it can not escape. Each attractor has a basin of attraction, that is the set of system states that lead to the attractor. There are two very basic type of attractors, point and cyclic. A point attractor is a steady state, once the system reaches the point attractor it does not change. The cyclic attractor is a set of states that are visited repeatedly in a loop.

The synchronization task can be defined as developing a transition table for the CA that produces the largest basin of attraction possible for the synchronized two-state cyclic attractor. Optimally, there would be only the synchronized cyclic attractor, covering the entire state space of the CA. Sub-optimal rule tables, like the hand-designed version presented above, have a great number of point and cyclic attractors that cover most of the state space. Figure 1 presents a simple cyclical attractor produced by a CA transition table that performed the synchronization task on a large number of random initial CAs. The attractor oscillates between two patterns, moving the pattern upward one step each oscillation.



Figure 1: A non-synchronized cyclic attractor in a high fitness CA transition table evolved by a GA

## 5 Parameters and Methods Affecting GA Performance

A great many factors affect the performance of a GA in any search task, synchronizing a CA being no exception. The fact that there are many malleable conditions inherent in CAs serves to further complicate the construction of a high performance search.

Simple GA parameters, such as mutation rate, crossover rate, and population size serve to balance exploration and exploitation while searching. The effects of varying these parameters were explored by the author in “The Effects of Parameter Variation on Genetic Algorithm Performance”.

In this case, it is the many parameters of the CA that are of interest.

## 5.1 Methods of Fitness Evaluation

The CA is used as the fitness function for the GA. Basically, if a genome representing the transition table of the CA leads to synchronization it has a high fitness. But, even in this simple statement there are many decisions to be made. Here are four ways that the CA can be used to test the fitness of any given genome, in increasing order for both complexity and accuracy:

1. One initial state of the CA is chosen at the beginning of the search. Each genome's fitness is determined by its ability to synchronize this CA after some number of iterations, say 300. So, the highest fitness would be all cells having state 0 at time 298, state 1 at time 299, and state 0 again at time 300 (or the inverse). Partial fitness can be given simply by  $\text{abs}(\text{sum}(\text{cells at time 298}) - \text{sum}(\text{cells at time 299})) + \text{abs}(\text{sum}(\text{cells at time 299}) - \text{sum}(\text{cells at time 300}))$ . The advantage of this method is that it is relatively easy and quick to execute. The disadvantage is that some genome may handle this specific initial state well but perform very poorly on most other initial states.
2. A new random initial state is given to each genome for every fitness evaluation. The ability to synchronize is given after some number of iterations as it was in method 1. The advantage here is that not all genomes are being evolved to handle one specific initial state, but rather any random initial state. One genome could be developed, however, that handled one random initial state well but others poorly.
3. A better approximation of fitness for each genome is obtained by testing it on several initially random CAs. Each CA is run for some number of iterations and the level of synchronization computed as before.
4. Further improvements in the accuracy of fitness evaluation may be obtained by testing the CA for synchronization throughout its execution. For example, the synchrony of times 100 and 101, 200 and 201, and 300 and 301 could be averaged. This results in higher fitnesses for CAs that synchronize sooner or are more nearly synchronized for longer periods of time.

The effects of testing more random initial states overall, more initial states per fitness evaluation, and at more points throughout the CA run all serve to produce a more accurate fitness evaluation but also a more complex and time consuming procedure. In the case of this research, fitness evaluation type 2 above is used because of its fairly substantial gain in generality over type 1 without much of the overhead of types 3 and 4 (running the CA and computing the sums is some of the most time consuming work).

## 5.2 CA Dimensions

The dimension of the CA used as a fitness measure has a large effect on the time required to compute the fitness as well as the time to find an individual genome that can perform the synchronization task. If very small CAs are used the GA will quickly find genomes that produce synchronization. However, these genomes perform very poorly if the dimensions are scaled up. If very large CAs are used it may take a very long time to find a perfect genome, but that genome will most likely be of higher quality. Square dimensions of 15, 20 and 30 were used in this case. Genomes that synchronized each of these dimensions were found by the GA.

## 5.3 CA Iterations

The number of iterations the CA is run over to compute fitness has a number of effects. Most obviously, the more iterations the longer the fitness evaluation takes and the fewer genome that can be tested in a given time period. Also, with more iterations the fitness of any given genome is likely to be higher. This is because it has longer to try and synchronize. Reducing the number of time steps allowed will lower the fitness but will allow for greater fidelity at the high-end. One possible approach is to start with a relatively high number of time steps, to find any genomes that are good, and gradually reduce the number of iterations, allowing further discrimination between high fitness genomes.

## 5.4 Initial CA Density

The initial density of the CA, the proportion of cells in state 0 to cells in state 1, will also have an effect on the time it takes to find high fitness genomes. To synchronize a CA it first must reach a condition where all cells are in state 1 or state 0. Then it is a trivial task of inverting the cells each time step. So, if the probability of a cell being in state 0 or 1 is skewed away from .5 the task will be easier. Finding the best transition table (that yields the largest basin of attraction for synchronization) requires that the hardest case be tested for, that is a .5 probability of any cell initially being in state 0 or 1. This is the approach used for these experiments.

## 6 Implementation

The GA and CA were initially implemented in MATLAB. The MATLAB implementation of the CA is included as Appendix A. The GA code is nearly the same as was used for an earlier project, “The Effects of Parameter Variation on Genetic Algorithm Performance”. In that project the fitness function was a simple sum of the bits in the genome. The CA used in this project is substantially more complex, and so took a great deal more processing time to do a fitness evaluation.

The MATLAB code was too slow to find CA transition tables that synchronized in a reasonable amount of time. It was very useful, however, in visualizing the progression of the CA over time. This allowed human inspection of the CAs and provided insight into the high-level methods the transition tables employ in the synchronization task.

To increase the performance of the search the system was also implemented in C++. This task was mainly undertaken by Samuel Inverso and is described in more detail in his paper, “Evolving Density Classifier Two-Dimensional Cellular Automata”. The performance increase in this implementation over that of MATLAB is an impressive factor of 50 or more. This high-performance version was used to find genomes that synchronized some random CA within 300 time steps. These genomes were then injected into the MATLAB version where they could be visualized. The progression of a 2-D CA is best done with a movie where the frames are the sequential states of the CA. The MATLAB code in Appendix A has this capability of displaying the CA during execution.

## 7 Results

Synchronizing CAs were found by the genetic algorithm for square dimensions of 15, 20, and 30. One of the best genomes found (able to synchronize the greatest number of initially random CAs) is:

```
11011001110111010001101110101011101110110111001111001100010011011110111010110011
0010011001101000110100101110011010000100010110101111110110110000001101101110101
111000011011101011010101110110101011011110000111011000011000000011110001000001
01001110110010001100100000110011111001000101101111011010110101001101110011001000
10101000101011011100100100000000001100011001000000001001101000111111110000111
00001110100101111110100001101011110010001100010011010001111101101011100100010010
01011100110101101110100000001000
```

In hex:

```
D9DD1BABBB73CC4DEEB32668D2E6845AFED81B75E1BAD5DAADE1D8603C414EC8C833E45BDAD4DCC8
A8ADC9000C640268FF870E97E86BC8C4D1F6B9125CD6E808
```

## 8 Future Work

The research presented here is merely a seed for a larger project being undertaken by the author and two colleagues, Samuel Inverso and Chadd Merrigan. That project will focus on a larger set of topics as well as making the results of experimentation more rigorous. Topics will include the 2-D CA synchronization task

presented here, as well as other 2-D CA tasks, such as density classification and a problem we have termed *balanced surface minimization*. This task aim to reduce the surface area between cells of state 0 and state 1 while maintain as close to a 50/50 balance in cells of the two states. So the optimal CA would come to an attractor where there are two large groups of cells, one with state 0 and one with state 1, with a straight line border between them.

## Appendix A – MATLAB Implementation of CA

```
function movie = ca ( initM, rules, timeSteps, showWhileRunning, pauseSteps )

% initM: the initial CA matrix
% rules: rules governing the state changes of each cell in the CA (all cells
%       have same rules). Each cell has a neighborhood of 9 (itself and 8
%       neighbors. The rules specify a target state (1 or 0) for each of the
%       2^9 possible neighborhood states.
%
% timeSteps: number of time steps to execute the CA for
% showWhileRunning: if true CA matrices are displayed as they are created
% pauseSteps: pause inbetween each step if = 1

% returns -- a 3D matrix representing the 2D world overtime. Can be played as a
%           movie by showing the layers in sequence

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ca.m
% Dan Kunkle
% Genetic Algorithms
% March, 2002
%
% A CA that begins with an initial matrix of cells (initM) and executes the
% given rules (rules) for timeSteps number of steps. CA is toridal.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

movie = initM;
[ rows, cols ] = size( initM );

if ( showWhileRunning )
    iptsetpref( 'ImshowTruesize', 'manual' );
    iptsetpref( 'ImshowBorder', 'tight' );
    f1 = figure( 'position', [ 100, 700, 600, 600 ] );
    set( f1, 'DoubleBuffer', 'on' );
end

% offsets in col and row direction from center cell to each of 9 neighbors
nIndexR = [ -1, -1, -1, 0, 0, 0, 1, 1, 1 ];
nIndexC = [ -1, 0, 1, -1, 0, 1, -1, 0, 1 ];

neighborsR = [];
neighborsC = [];
% build neighbor table
for r = 1:rows
    for c = 1:cols
        for j = 1:9
            neighborsR( r, c, j ) = mod( r + nIndexR(j) - 1, rows ) + 1;
            neighborsC( r, c, j ) = mod( c + nIndexC(j) - 1, cols ) + 1;
        end
    end
end

for i = 1:timeSteps
```

```

if ( showWhileRunning )
    imshow( movie(:, :, i) );
    pause(.001);
    if ( pauseSteps )
        pause;
    end
end

% update matrix based on rules

% for each cell...
for r = 1:rows
    for c = 1:cols

        % find neighborhood state (use mod to create torus)
        nState = [];
        for j = 1:9
            nState(j) = movie( neighborsR( r, c, j ), ...
                neighborsC( r, c, j ), i );
        end

        % find rule corresponding to neighborhood state and
        % set cell state
        movie( r, c, i+1 ) = rules( bin2dec9( nState ) + 1 );

    end
end

end

```

## References

- [1] Rajarshi Das, James P. Crutchfield, Melanie Mitchell, and James E. Hanson. Evolving globally synchronized cellular automata. In Larry Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 336–343, San Francisco, CA, 1995. Morgan Kaufmann.