MatrixCA: A Game of Life Generalization

Jason Rohrer, Eric Shulman, and Stefan Gross

Created: February 6, 2002; Last modified: February 6, 2002

1 Introduction

With MatrixCA, we have tried to generalize some of the principles behind Conway's original Game of Life (GOL). The GOL rules operate only on a binary domain and range. Our generalization can operate on domains and ranges taken from anywhere on the real line, though the special cases that we present only make use of subsets of the integers.

When designing this system, we sought to make it suitably powerful, but we also wanted to ensure that the GOL was a special case of our generalization. In fact, generalizing from the GOL adds power to our system, since the GOL is a Turing-complete computation system. Thus, by reduction from the GOL, our system is also Turing-complete.

2 Generalization

Our generalization is composed of three parts: a discrete grid of real-valued cells, a weighted neighborhood summation matrix, and a filter function.

We use a wrap-around grid of discrete cells for simulating our rules. Each cell assumes a single real value per time step. During each time step, new values for each cell are computed simultaneously for all cells using the values from the last time step.

The summation neighborhood around a cell is a real-valued $n \times m$ matrix, where both n and m are odd. The neighborhood matrix is fixed for all cells in the grid. The matrix can be thought of as a set of weights on neighbors' values (and its own value) that a cell uses to compute its value for the next time step. For example, consider the following matrix:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

For a given cell, this matrix specifies an output that is the sum of all of its direct neighbors' values plus two times the cell's own value.

The output of the matrix sum for a given cell is fed into a filter function, and the output of the function specifies the cell's value at the next time step. The filter function can be any real-valued function. The function is fixed for all cells in the grid. For example, consider the function

$$f(x) = \lfloor x \rfloor \bmod 5. \tag{1}$$

A graph of this function can be seen in Figure 1.

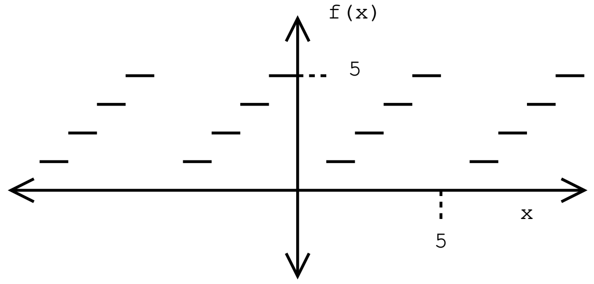

Figure 1: The modulo-5 function.

Using these two rule elements, summation matrices and filter functions, we have a very general system. Simple matrices and functions can be specified to encode the rules of the standard GOL, as described below. We believe that all GOL variants are special cases of our system, though certain rule sets require more functional acrobatics than others. Furthermore, we have a very rich parameter space that extends well beyond the bounds of simple GOL variants.

3 Special case examples

3.1 Conway's game of life

The standard GOL rules can be encoded in our generalization in the following way. First, we use the following matrix:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -10 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{2}$$

Next, the sum produced by this matrix for each cell is passed through the following function:

$$f(x) = \begin{cases} 1 & \text{if } \lfloor x \rfloor = 3, -7, -8 \\ 0 & \text{otherwise.} \end{cases}$$

A graph of this function can be seen in Figure 2 on page 3. For the sake of clarity, let 0="dead" and 1="live". If a dead cell is surrounded by exactly three live neighbors, the sum of the matrix is three, and the cell becomes live. If a live cell is surrounded by two or three live neighbors, then the sum of the matrix is -10 + $\{2 \text{ or } 3\}$, or $\{-8 \text{ or } -7\}$, so the cell stays alive. Otherwise, the cell becomes (or remains) dead.

This encoding of the GOL rules produces exactly the same behavior as the standard GOL rule mechanisms (*i.e.*, birth and death rules based on neighbor count and a cell's current state).

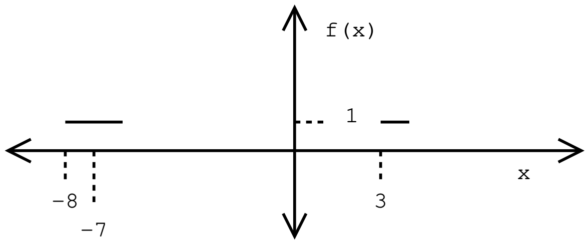

Figure 2: The GOL filter function.

3.2 Modulo filter functions

Another interesting class of special cases uses modulo filter functions. One such function (with modulus 5) is described in Equation 1 on page 1 and graphed in Figure 1 on page 2. Any neighborhood summation matrix can be used, though symmetric matrices seem to produce the most interesting effects.

Many of our experiments involved a modulo-11 filter function, along with the standard GOL matrix (as seen in Equation 2 on page 2). We simulated these rules on a 40×40 wrap-around grid. With this particular configuration, a fascinating phenomenon occurs. No matter what start state is entered into the grid, that state is reached again by generation 120.

Even more interesting are the intermediary states (generations 2 through 119) during this cycling process: they are comprised of an amazing variety of intricate patterns. Depending on the initial state, these patterns can range from symmetric mandala-like structures to structures that are indistinguishable in appearance from random noise. However, no matter how complex the intermediate states are, the start state reappears in generation 120. Furthermore, every eleven generations (generation 11, 22, 33, *etc.*), structures that somewhat resemble the original state appear. However, this 11-generation pseudo-cycle is broken for the 121st generation, since the initial state is hit in generation 120, *not* generation 121. Note also that this type of behavior seems tied to the 40×40 grid, since the behavior changes for other grid sizes.

Similar behavior can be found with other moduli that are multiples of 11. Also, by changing the matrix somewhat, cycling behavior can also occur for moduli that are not multiples of 11. For example, by replacing the -10 with a 1 in the GOL matrix and using a modulus of 2 (binary cell states), we get cycling after 24 generations.

3.3 C-F neighbor interactions

The C-F special case, which is described in more detail in Section 5, tries to generalize the GOL rule paradigm to non-binary cell values. This generalization involves incrementing or decrementing a cell depending on the sum of its neighbors. Fitting the standard GOL rules into our generalization required a bit of functional acrobatics, as seen above, but the function corresponding to the C-F rules is even more complex.

Let us assume that M is the maximum cell value allowed, and that the minimum cell value is 0. First, we use the following 3×3 matrix to produce a weighted sum the cell and its neighborhood:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 18M & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{3}$$

We want to pass this sum through a function that can somehow differentiate between various different center cell values so that they can be incremented or decremented appropriately. The above matrix summation will spread the sums out: we will have M distinct sum clusters along the real line, one corresponding to each of M possible center cell values, and the centers of these clusters will be separated from each other by 18M.

Consider the case where $M = 2$. Our matrix becomes

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 36 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{4}$$

For each possible center cell value, 0, 1, or 2, we have a cluster of matrix sum points. Consider the case where the center cell has a value of 0. This cell's cluster of possible matrix neighborhood sums starts at 0 (when all neighbors are 0) and ends at 16 (when all neighbors are 2). If the center cell has a value of 1, its sum cluster starts at 36 and ends at 52. Note that the end of the 0-cluster and the start of the 1-cluster are separated by 20. This separation is the basis for a filter function that can differentiate between various center cell values.

Let $f'(x)$ be the pure increment-decrement function that we would want to apply for a cell given the non-weighted sum of it and its neighborhood. For example, if $M = 2$ as before, $f'(x)$ might be the step function graphed in Figure 3. With this function, if the

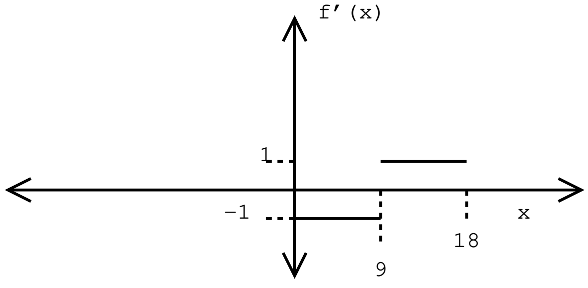

Figure 3: A step increment-decrement function.

non-weighted sum of the neighborhood is in $[0, 9)$, we decrement the center cell's value, and if the sum is in $[9, 18]$, we increment the center cell's value.

We want to embed this increment-decrement function into a well-defined univariable function, $f(x)$, that will actually increment and decrement the center cell values give the sums from the matrix in Equation 4. This can be achieved with the following function definition:

$$f(x) = \begin{cases} f'(x) + 0 & \text{if } x \in [0, 18M) \\ f'(x - 18M + 1) + 1 & \text{if } x \in [18M, 36M) \\ f'(x - 36M + 2) + 2 & \text{if } x \in [36M, 54M) \\ f'(x - 54M + 3) + 3 & \text{if } x \in [54M, 72M) \\ \vdots & \\ f'(x - y * 18M + y) + y & \text{if } x \in [y * 18M, y * 18M + 18M) \\ \vdots & \\ f'(x - 18M^2 + M) + M & \text{if } x \in [18M^2, 18M^2 + 18M) \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

In the case where $M = 2$, we get the following function:

$$f(x) = \begin{cases} f'(x) + 0 & \text{if } x \in [0, 36) \\ f'(x - 36 + 1) + 1 & \text{if } x \in [36, 72) \\ f'(x - 72 + 2) + 2 & \text{if } x \in [72, 108) \\ 0 & \text{otherwise.} \end{cases}$$

If $f'(x)$ is the step function graphed in Figure 3 on page 4, then we produce the $f(x)$ graphed in Figure 4.

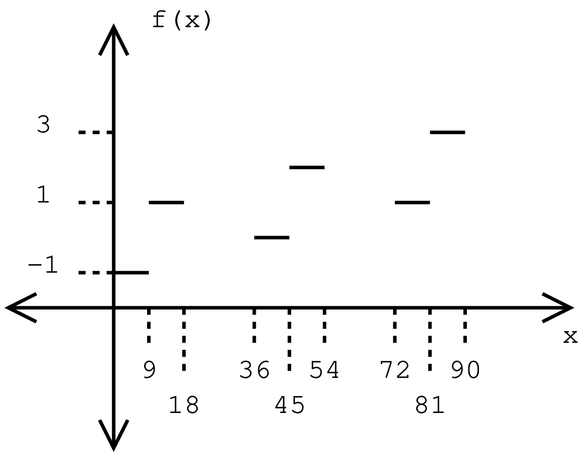

Figure 4: A C-F filter function for $M = 2$ and the step function from Figure 3 .

Notice that there is still a piece missing from our encoding: we mentioned before that cell values must be in the range $[0, M]$. The function $f(x)$ as shown Figure 4 does not preserve this property in its range. We actually need to wrap $f(x)$ with another super-function $\bar{f}(x)$:

$$\bar{f}(x) = \begin{cases} x & \text{if } x \in [0, M] \\ M & \text{if } x \in (M, +\infty) \\ 0 & \text{if } x \in (-\infty, 0) \end{cases}$$

Our final function, $f^*(x)$, thus becomes

$$f^*(x) = \begin{cases} \bar{f}(f'(x - y * 18M + y) + y) & \text{if } x \in [y * 18M, y * 18M + 18M) \\ & \text{for some } y \in \{0, 1, \cdots, M\} \\ 0 & \text{otherwise.} \end{cases}$$

We have shown that we can encode the value of the center cell *and* the pure neighborhood sum in a single weighted matrix sum. Thus, any cellular automata rule-set that depends on the neighborhood sum and the current cell value is a special case of our system. Note that this generalization works for *any* increment-decrement function $f'(x)$. In the actual C-F special case described below, $f'(x)$ is an inverted parabola.

The GOL special case described in Section 3.1 is a simple application of the ideas described here. However, we should note that the increment-decrement function $f'(x)$ for the GOL special case is actually further parameterized by the current cell value (to differentiate between the birth rule and the continued life rules). This can be achieved by using a different $f'(x)$ for each sub-case of Equation 5. We still have a well-defined, univariable function.

4 Parameter exploration tool

We have developed a Java tool for exploring the parameter space of MatrixCA. The distribution for this tool can be downloaded from:

`http://gca.sourceforge.net/matrixCA`

The MatrixCA GUI features two modes: the generic 3×3 mode and the C-F mode. While in the 3×3 mode, the matrix entries can be manipulated in real-time as the simulation progresses. In the C-F mode, the three parameters that control the C-F rules can be adjusted in real-time.

While in either mode, the modulus of the global modulo filter function can be adjusted. The GUI also provides a point-and-click interface for adding in-domain values to the grid. Generation step size is also adjustable, so many generations can be passed through quickly without the overhead of updating the display at every generation.

5 C-F in detail

In Conway's Game of Life, a cell will die (or remain empty) if it is surrounded by too few or too many live cells, and will be active when "supported" by the right amount of neighbors. The algorithm for C-F basically extends this paradigm to continuous-valued cells: at each time step, for each cell, the sum of the values of its neighbors is passed through an inverted parabolic function. If too low or too high, the parabola returns a negative value, and the parabola's maximum is situated in the middle, where the value goes above 0. In both cases the resulting value is then added to that of the current cell for the next time step.

The formula used to compute this change in value is

$$\Delta(cur, val) = \frac{-(val - crowd)^2}{fitness * (maxhealth - cur)} + maxchange$$

where **cur** is the value of the current cell, and **val** is the sum of the neighboring values. **crowd**, **fitness**, **maxhealth** and **maxchange** are modifiable parameters which we will describe shortly.

$\Delta(cur, val)$ is capped at **-maxchange**, and obviously won't go above **maxchange**. The closer to the **crowd** value the current cell is, the closer to the maximum of the parabola. This is set to match the Game of Life's situations where a cell will thrive if it has a number of neighbors close to 2.7, and die off otherwise. The width of the parabola is determined by the **fitness * (maxhealth - cur)** component: higher-valued cells will find it harder to have a compatible neighboring configuration.

The different parameters can be modified through the GUI to generate various "world physics":

- The **maxhealth** value is set via the Max field, which determines the maximum value of a cell; possible values are capped between 0 and this maximum value. This basically represents the energy store a cell can have.

- C represents the **crowd** value: lower values favor the formation of new cells, and thus yields more instability, while higher values tend to allow already existing structures to thrive. Basically this parameter determines how far to the right of the y-axis the parabola's maximum is.

- F represents the **fitness** of cells, how likely a cell is to grow the further the sum of its neighbors gets from the parabola's maximum.

- scale represents the **maxchange**, and determines by how much the value of a cell can rise or fall (within the boundaries set with Max); the magnitude of value the parabola returns is bounded by this parameter.

Interesting parameters to try (be sure the Matrix Type is set to C-F):

- Max=30 C=25 F=0.8 sc=5: with these, the CA will eventually create a network of roads with some internal motion, when started with a single cell or with a randomized board.

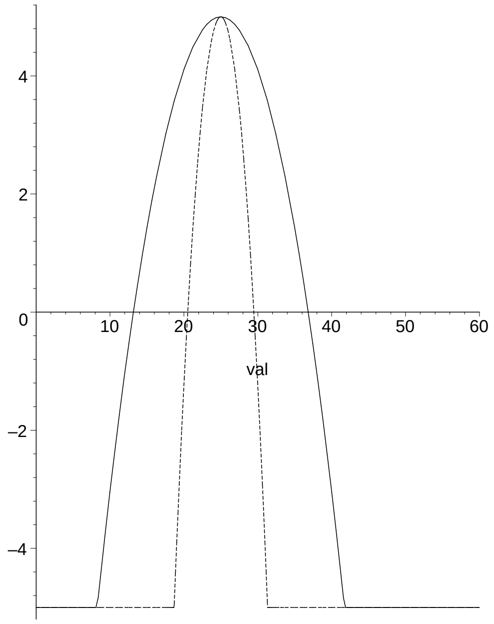

Figure 5: Output of $\Delta(cur, val)$ when **cur** = 0 (solid line) and 30 (dashed line).

- Max=20 C=20 F=0.18 sc=5: starting from a randomization, or from a sufficiently large group of level 20 cells, a kind of flocking behavior emerges: groups of cells will seem to move in random directions, spawning subgroups which may die out or evolve into new migrating groups.

- Max=10 C=58 F=25 sc=2: starting from a sufficiently large group of active cells, this will form a fairly stable group, which will attempt to regain its form when part of it is removed or it is cut in half (while the simulation is temporarily stopped).

- Max=20 C=35 F=1 sc=7: ends up with another network of roads, this time with roads constantly being formed and then erased little by little.

- Max=20 C=30 F=0.8 sc=10: generates groups of horizontal and vertical lines; the groups vie for control of the whole grid, and after a while one of the groups prevails.