A Particle Swarm Selects for Evolution of Gliders in Non-Uniform 2D Cellular Automata

JJ Ventrella

Jeffrey@Ventrella.com

Simulation online at: http://www.ventrella.com/Alife/Cells/GlidersAndRiders/Cells.html

Abstract

Coherent space-time structures (gliders) which emerge in class IV cellular automata (CA) can be seen as "vehicles" that move information, as they are common in CA which support universal computation. A technique for evolving gliders from heterogeneous 2D CA is described. Rather than use a genetic algorithm on a population of rules, and image-filtering to detect structures for measuring fitness, a particle swarm is employed which interacts intimately with the CA and performs genetic operators locally on the heterogeneous rules, as the dynamics emerge. The swarm selects for coherent motion. These particles do not fly on a search mission – instead, they "ride" on the backs of clusters of emerging structures, due to attractive forces. In exchange for a "good ride", they reward local dynamics with more coherent motion by performing genetic operators of selection and reproduction. This technique not only demonstrates an efficient way to evolve a large variety of gliders: it also simulates emergent complexity through co-adaptation.

Introduction

Cellular automata (CA) are discrete dynamical systems used to explore general principles in terms of how complexity arises from simple rules. Much research using CA is centered on universal computation – the ability for CA to generate dynamics representing basic logical operators that are the foundation of computation, as in a Turing Machine. Research in evolving CA rules that exhibit universality have implications for theories on the origins of life. As put by Chris Langton: "In living systems, a dynamics of information has gained control over the dynamics of energy, which determines the behavior of most non-living systems. How has this domestication of the brawn of energy to the will of information come to pass?" (Langton, 1992).

CA that support universality are of class IV dynamics (Wolfram, 1984). This is the class of dynamics supporting coherent space-time structures, known as gliders. Gliders, as well as various kinds of waves and multi-glider configurations, are structures that propagate across the CA space against a quiescent or periodic background. They can be described as "vehicles" which move signals across the CA space. It has been shown that the interactions of multiple gliders in controlled arrangements in the *Life* Universe, can represent the basic logical operators of computation. (Smith 1971), (Berlekamp, et al. 1982), (Rendell 2000), and others, have developed CA's capable of universal computation.

Variations of the Genetic Algorithm (GA) have been used to evolve CA rules which support dynamics exhibiting universality. (Das et al. 1994) developed GA's to evolve populations of CA rules. The resulting dynamics of their research are able to perform computational tasks. (Wuensche, 2001) has developed ways to automatically detect space-time patterns in CA, and has classified many kinds of dynamics.

This paper proposes a technique which is *symbiotic*, and *emergent*: a collective of particles interact with a 2D CA lattice with heterogeneous rules (transition functions). These particles detect coherent motion within the CA dynamics, communicate with each other about it, and – by performing genetic operators asynchronously over time – actually promote the evolution of gliders (defined in this paper as structures that propagate in a coherent direction and speed against a quiescent background – and have long life spans.)

The paper is organized as follows: First is an explanation of why a swarm-based technique was chosen over a visual/analytical approach. Then, comparisons to the standard particle swarm optimization technique are given. The CA model is then described, followed by a description of how the particles apply the genetic operators on the CA. An explanation of the parameters is given, and finally, observations from experiments are given.

Visual *Observer* vs. Interactive Agent

The eye/brain system can easily detect many kinds of motion, and so we are able to recognize various moving patterns in CA. This may be one reason why Conway's *Game of Life* (Gardner, 1970) is so mesmerizing to watch. (In this paper, Conway's Game of Life is simply called, "*Conway*").

In a previous exploration in evolving gliders, an interactive evolution interface was developed and published online (Ventrella, 2000). With repeated viewings of the dynamics of individuals from a population of transition functions, the user judges according to aesthetics. The user's choices act as the fitness function for a genetic algorithm. The question of how to automatically select for gliders was considered, and this lead to the current project. An earlier version of the technique presented here used a convolution filter to identify fuzzy objects which persist over time. This proved to be CPU-

intensive, especially when applied over simulated time. But more importantly, the conclusion was made that it is not germane to the subject of interest: emergent complexity – how an information-dynamic may have emerged from within inanimate matter. Using a *visual* model seemed inappropriate.

The work of (Ramos and Almeida, 2000) in modeling artificial ant colonies to detect features in digital images, using pheromone modeling, provided motivation for taking a *swarm* approach. Such techniques use collaborative filtering, and demonstrate *stigmergy*, a principle introduced by (Grassé, 1959) to explain some of his observations of termite nest-building behaviors. Stigmergy recognizes the environment as a stimulus factor in the behavior of an organic collective, which in turn affects that environment.

Particle systems traditionally used in computer graphics (Reeves, 1983) provided further inspiration for some of the techniques used involving physical simulation.

Particle Swarm Guides Evolution

Particle Swarm Optimization (PSO) (Kennedy and Eberhart, 1995) is a searching and optimizing technique that is effective across many problem domains. It is modeled after a kind of collective adaptive behavior, demonstrated by the social systems of humans, insects, and other living systems comprised of many interacting parts. This paper introduces a technique which has similarities with PSO, but departs in a few significant ways. A collection of particles is superimposed upon the CA lattice, as illustrated in Figure 1. At every time step, a particle occupies the area corresponding to a unique cell in the CA lattice (imagine a marble rolling across a square-tiled floor). Particles are attracted to "live" cells, and can detect them if they exist within a local neighborhood.

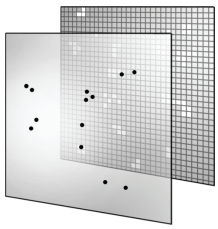

Figure 1: The particle swarm maps to the CA space

These live cells generate an attraction force which is applied to the particle's velocity. The quality of the particle's velocity is then subject to interpretation by the particle itself, based on a selection criterion *S*. When a superior quality is found, the particle stores the genes from the associated local area in the CA. (analogous to *pBest* value in PSO). Copies of the selected genes are continually deposited back into the CA over time. When two particles come in contact with each other, the particle with the highest value gives a copy of its genes to the other particle. This local, social exchange reinforces the effect of selection. There is no calculation of global best (analogous to *gBest* in PSO). Soon after initialization, local regions of relatively homogeneous genetics emerge. Some of these are dense and highly chaotic, and effectively "trap" particles. Other regions give birth to short-lived propagating structures, or "proto-gliders". These proto-gliders can transport the particles across greater distances, which then spread further the genes they have selected. In most simulation runs, a secondary phase takes over in which new gliders expand the genetic regions from which they originated. This phase usually results in a rapid take-over by one transition function supporting gliders.

The CA

The 2D lattice of automata are arranged in a square grid. Periodic boundary conditions enable the dynamics to wrap-around. Time is measured in discrete steps. A cell can assume any state in the range from 0 to the number of possible states *K*. State 0 is the *quiescent* state. At each time step the state of each cell can be changed to another state according to that cell's transition function.

As mentioned before, each cell has its own unique transition function. The 9-cell Moore neighborhood is used as a local environment to determine the next state of the cell at each time step *t*. The transition function consists of *R* "sub-rules" applied in sequence. A single sub-rule is expressed as follows: the cell's current state is compared to a reference state (the *referenceState*). If the cell's state does not match *referenceState*, then the sub-rule is not applied, and the cell defaults to quiescence. Otherwise, the sub-rule compares the number of neighbors having a specific state (the *neighborState*) to a specific number (the *neighborCount*). If there is a match, then the sub-rule sets the cell's new state to a specific result state (the *resultState*). Thus, four parameters are used for each sub-rule, which are *genetic*, i.e., they can have a range of possible values, and can be changed by a genetic operator. The possible ranges are as follows:

1. *referenceState* can be any state in *K*
2. *neighborState* can be any in *K* state except quiescent
3. *neighborCount* can be any number from 1 to *M*
4. *resultState* can be any state in *K*

K has been tested with values as high as 20. The number of sub-rules *R* has been tested with values as high as 80, and thus the number of genes per transition function 4*R* can be as high as 320. The set of all genes for a transition function is referred to as the *gene array*. The pseudocode below illustrates how the sub-rules are applied to determine the value of the new cell state.

```
SET new cell state to quiescent

FOR each sub-rule
   IF current cell state EQUALS sub-rule.referenceState THEN
      IF number of neighbors of state sub-rule.neighborState
      EQUALS sub-rule.neighborCount THEN
         SET new cell state to sub-rule.resultState
      ENDIF
   ENDIF
END FOR
```

To place this in a familiar context: the transition function would require three separate sub-rules to define *Conway*, and the gene values would be as follows:

sub -rule#	cell -state	neighbor -state	neighbor -count	result -state
1	0,	1,	3,	1
2	1,	1,	2,	1
3	1,	1,	3,	1

The number of genes required for *Conway* is 12. Of course the essence of *Conway* can be expressed with fewer parameters, using more elegant notations, but the purpose of this experiment is not elegance.

Given a large number of sub-rules, and the initial set of random values, it is often the case that some of these sub-rules are redundant – having the exact same values. Furthermore, it is likely that many of these sub-rules, as they are applied in sequence, will "over-write" the results of previously applied sub-rules. From a Computer Science point of view, this would be considered sloppy programming. However, this encoding of the CA transition function allows for a larger, more flexible genetic space for experimentation.

The Particles

The particle swarm consists of n (typically 500) particles. Unlike the cells of the CA, the particles occupy a real number space (normalized to a square of size 1), and can thus move continuously. Similar to the CA, particles have periodic boundary conditions, and so if a particle falls off an edge of the domain, it re-appears at the opposite edge, maintaining its velocity. Velocity and position are updated every time step t as follows: For the ith particle...

$$\mathbf{v}_i(t) = \mathbf{v}_i(t-1)\omega + \mathbf{a}_i + \mathbf{r}_i$$
$$\mathbf{p}_i(t) = \mathbf{p}_i(t-1) + \mathbf{v}_i(t)$$

$i = \{1, 2...n\}$, where n is the number of particles. **p** is position, **v** is velocity, and ω is the inertia weight ($0 \leq \omega \leq 1$). Two forces: **a** and **r**, are added to the velocity at every time step, as explained below.

To keep the particles from going out of control as forces **a** and **r** are added to **v** over time, **v** is scaled at each time step by an inertia weight ω. (Shi and Eberhart, 1998) introduced an inertia weight to the standard PSO technique, which affects the nature of a particle's "flying" behavior. The particles in this scheme perform different tasks than in PSO (they don't actively *fly*, they passively *ride*). In either case, the precise tuning of the ω is important, as explained below.

The Particle Cell-Neighborhood

At every time step a particle maps to a unique cell in the space of CA, called its "reference cell". The particle continually reads the contents within a local Moore neighborhood of radius 2 (number of cells PN = 5X5 = 25), which surrounds and includes the reference cell, as shown in Figure 2. (Note that this neighborhood is larger than the neighborhood for the CA transition function). If one or more of the cells in the neighborhood are non-quiescent, the velocity of the particle is accelerated by an attraction force **a**, which is the sum of all vectors from the reference cell to each non-quiescent cell in the neighborhood. This vector is normalized and scaled by the attraction weight aw.

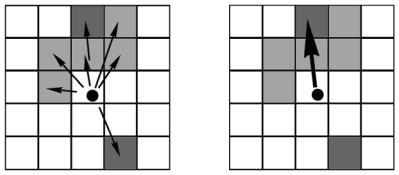

Figure 2: A particle cell-neighborhood containing 7 non-quiescent cells. At left are the 7 resulting attraction vectors. At right is the vector resulting from summing these vectors and normalizing the sum

When a particle's velocity **v** is being accelerated by force **a**, the particle is said to be "riding", and the "quality" of the ride q ($0 \leq q \leq 1$) is determined by the selection criterion S, which is a function of velocity over time. In most experiments, the criterion is configured so as to favor a constant velocity and a maximum speed. This is explained below. When the particle is <u>not</u> riding, a random force vector, **r**, is added at each time step. It has magnitude rw (random weight), and one of four possible orthogonal directions, randomly chosen at each time step. This force causes the particle to meander in a Brownian fashion, and allows momentary dense clumps of particles to dissipate. Higher values of rw cause faster dissipation.

The Genetic Operators

Every particle has a random chance of dying and being re-born, at a rate of b ($0 \leq b \leq 1$) chance per time step. When a particle is re-born, its position is reset to a new random location and it takes a copy of the gene array from its new reference cell. Particles perform four genetic operators, illustrated in Figure 3, and explained below.

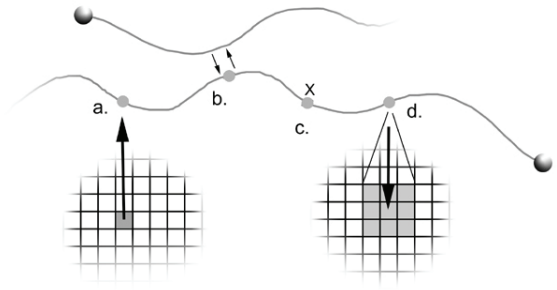

Figure 3: The four genetic operators: (a) selection, (b) exchange, (c) mutation, and (d) reproduction

a. Selection
When a particle experiences "the best ride in its life" (the best *q* found so far: *bq*), it stores this along with the genes from its reference cell. If at any time a better *q* is found, it stores this as *bq* and again stores the genes from its current reference cell. This is analogous to the "pBest" value used in PSO. Effectively it is a fitness metric and the genotype associated with that fitness.

b. Exchanging Genes
Every particle has a random chance per time step *e* ($0 \le e \le 1$) of exchanging genes with another particle (if it comes in contact with another particle: distance between the particles is less than one cell width). When an exchange occurs, a copy of the genes of the particle with the higher *bq* value replaces the genes of the particle with the lower *bq* value. This provides an extra level of reproduction, taking advantage of swarm intelligence to optimize the search. In PSO, the best value within a local particle neighborhood is typically called *lBest*. While no such value exists in this method, this social exchange effectively encourages a "local best" to emerge more rapidly than otherwise.

c. Mutation
Each particle has a random chance of genetic mutation, at a rate of *m* per time step ($0 \le m \le 1$). When a mutation occurs, one of the genes in its array is reset to a random value within its full range.

d. Reproduction
Particles randomly deposit random portions of their gene arrays into the CA. The rate of depositing is *d* ($0 \le d \le 1$) random chance per time step. This operator uses a variation on the standard GA crossover except that it's "one-way": the genetic information originates entirely from the particle, and is transferred to the particle's cell neighborhood, as illustrated in Figure 4.

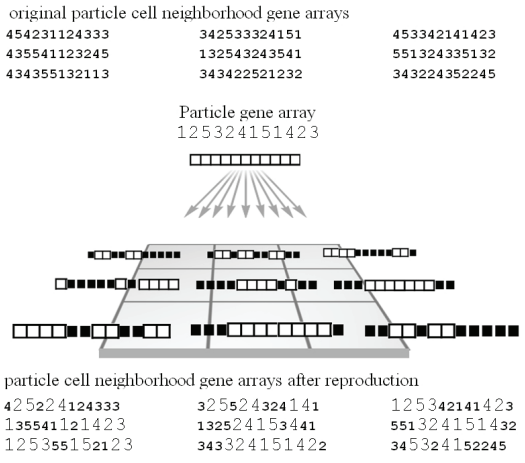

Figure 4: An illustration of how a particle deposits portions of its gene array into the neighborhood of its reference cell, using a "one-way" version of genetic crossover.

The effect of this operator is that approximately one half of the genetic information originating from the particle is spread out into a local region. Repeated reproduction operations in the same region with similar genetic information increases genetic homogeneity in that region. Crossover also effectively introduces a form of sexual reproduction, and allows potentially successful new genotypes to emerge from the combination of gene array building blocks.

Parameters

Various runs of the simulation were tried with different parameter settings. All variations are based around a standard configuration, shown below:

For the CA:
- number of cells in both dimensions (*resolution*) = 128.
- number of cell states $K = 4$
- number of transition function sub-rules, $R = 16$
- maximum neighbor count $M = 5$

For the Particle Swarm:
- number of particles $n = 500$

Parameters Affecting Particle Motion:
- particle cell-neighborhood number of cells PN = (5X5 = 25)
- inertia weight $\omega = 0.7$
- attraction weight $aw = 0.25$ / *resolution*
- random weight $rw = 0.13$ / *resolution*

Parameters for the Genetic Operators:
- mutation rate m = 0.0001 random chance / time step
- rate of depositing genes $d =$ 0.05 random chance/time step
- crossover rate $c = 0.3$
- rate of genetic exchange per particle (if there is another particle within one cell's distance) $e = 0.1$ random chance/time step
- rate of re-birth $b = 0.001$ random chance/time step

The selection criterion *S* is set to "constant and fast", favoring non-quiescent structures that move at a constant velocity, at the speed of light. Its measurement, the ride quality per time step, *q*, is defined as...

$$q_i = (|\mathbf{v}_i| - |\mathbf{d}_i|) / C$$

where **v** equals velocity; **d** equals the change in velocity since the previous time step; and *C* equals the speed of light (equal to the width of one cell). With this selection criterion, if particle i is riding a glider which is moving at the speed of light, then q_i will approach 1. Given the usual settings for ω and *aw*, $|\mathbf{v}_i|$ is rarely > *C*, and $|\mathbf{d}_i|$ is rarely > $|\mathbf{v}_i|$. Still, *q* is clamped: ($0 \le q \le 1$).

Adjusting Parameters

Changing the parameters associated with the CA has little effect on particle performance. Higher values of K and R cause more complex dynamics, often with exotic gliders and waves.

The parameters affecting motion, specifically ω and aw, are important for the particle's ability to generally track coherent motion. For instance, if aw is set too low, the particle will not experience enough attraction to live cells. If it is too high, the motion will be too erratic. Setting ω higher has the effect of smoothing out a trajectory which would otherwise be erratic due to small changes in the particle's cell-neighborhood: some gliders change their shapes as they migrate across the CA space – larger ω allows the particle to respond less to these small changes and more to the overall position and velocity. Low aw values and high ω values cause the effect of a low-pass filter on the trajectory. Changes to these two parameters, as well as the size of PN, allow different qualities of CA dynamics to affect particle motion.

The settings for crossover rate and mutation rate have similar effects on evolution as in the case of the standard GA. For instance, not enough mutation causes premature convergence to a local optimum, while too much degenerates into a random walk.

The parameters, n, d, e, rw, and b are more unique to this particular technique – they affect the rates at which the particles perform the genetic operations. The standard setting shown above is a good configuration – but it could probably be tuned precisely for more optimal performance.

Stirring the Primordial Soup

All simulations begin with total quiescence. The only stimulus used in this simulation is a linear "noise wave" of random non-quiescent cell states which immediately sweeps across the entire CA domain at a constant rate. The width of the wave is equal to the width of 1 cell. Density w is set to = 0.2 in most simulations (a density of 0 has no effect, and a density of 1 creates a solid wave of non-quiescent cells).

The wave repeats its sweep every W = 500 steps. The wave moves at a speed of $2C$ (the sweep lasts *resolution*/2 time steps). Moving faster than C reduces interference with the resulting dynamics – which can never have features that propagate faster than C. The noise wave remains dormant between sweeps. This period of dormancy allows the dynamics to settle to its native behavior.

The CA can sometimes be in a highly chaotic state – either because R is high and the transition functions are generating a high proportion of non-quiescent cells, or a chaotic transition function has temporarily dominated the CA). In this case, the noise wave is not necessary, as the soup is effectively stirring itself. However, the dynamics inevitably shift towards the *edge of chaos*: it approaches that critical boundary in which too much order is just over the edge, and the dynamics can easily slip into quiescence in large regions of the CA. The noise wave in this case helps to provide some momentarily stimulation.

Observations

Figure 5 shows the output of two simulations. At the left is a graph plotting the average ride quality of all particles *which are riding* (*aq*). At the right is a snapshot of the resulting CA dynamics.

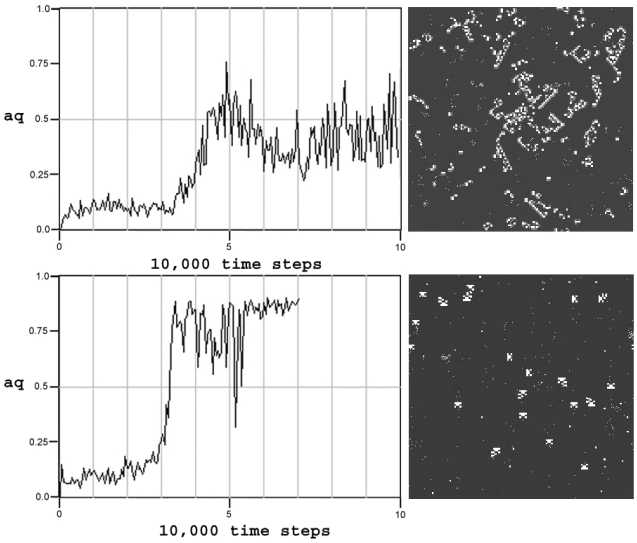

Figure 5: Output from two simulations. Average ride quality over time shown at left; snapshot of final CA dynamics at right.

The second simulation was run for approximately 70,000 time steps. In both of these examples, a sudden jump in *aq* can be seen. This is attributed to the emergence of a local homogeneous region of transition functions which produced superior gliders.

Figure 6 shows the results of a simulation in which R was increased to 32 to encourage more complex transition functions. At around time 25,000 (Figure 6a) solid, long, and narrow structures become dominant. At around time 65,000 (Figure 6b) the solid structures have broken into individual gliders which are emanating rapidly from small localized chaotic regions, in opposite directions (indicated by the arrows). These chaotic regions cause a sort of nucleation for growth, and act as *glider guns*.

Because the structures seen in 6b are separated by quiescent gaps, the particles are able to detect a better ride, due to more coherent attraction forces originating from variance in their cell-neighborhoods. Note the increase in *aq* at around 60,000 in the graph (Figure 6d). By time 100,000 (Figure 6c), a uniform global dynamic has taken over – and the final scenario consists of a few small gliders

Figure 6e shows a close up of the box drawn in Figure 6c. A glider can be seen moving upward. It has recently passed through a cluster of particles, many of which have "caught a ride" on the glider because they were located in its path. Some of these particles were only temporarily propelled by the passing glider, and are consequently left trailing behind, while others can be seen riding on the back end of the glider. The gliders in this CA are smaller than the average glider. Larger gliders more easily collect

particles because they occupy more non-quiescent cells in the particle's cell-neighborhood.

The erratic jumps in *aq* in the graph between time 60,000 and 90,000 are most likely the result of sparse data sampling since there is so much quiescence and fewer riding particles, and the CA has not yet settled into its final state. By around time 90,000, the gliders are few and sampling is still sparse, but collisions are infrequent.

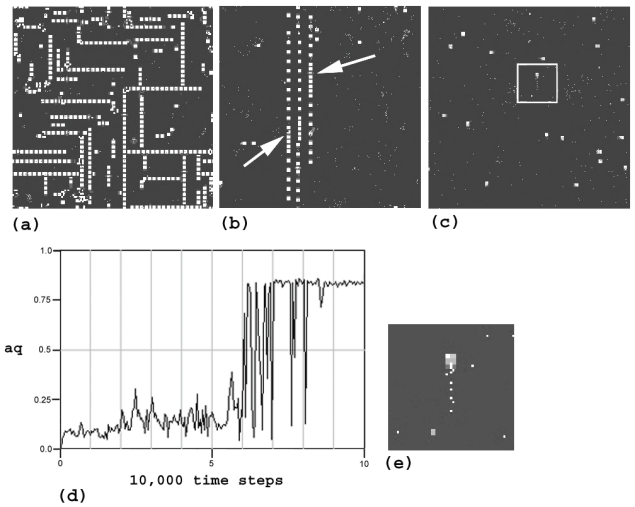

Figure 6: Three stages in the evolution of a glider-rich CA

Evolving *Conway* at λ = 0.273

The lambda value λ ($0 \leq \lambda \leq 1$) of a cell is defined as the amount of non-quiescence resulting from its transition function. This value is calculated using a function which generates every possible neighborhood state given *K* and *R*, and applies the transition function on each of these neighborhood states. The sum of all non-quiescent values of *resultState* from applying the transition function is determined, and this is divided by the total number of results.

A number of simulations were run which included this calculation. Lambda values are shown in Figure 7 as a scatter plot in the same graph as the value of *aq*. (Since both values fall within the range 0-1, they can be mapped to the same graphical window). The scatter plot is generated by measuring λ from 10 randomly selected cells at each time step, and then plotting these 10 values as dots. These simulations were run for 200,000 time steps. The particles are not displayed in the image of the resulting CA dynamics.

Langton (Langton 1992) found the λ for *Conway* to be approximately 0.273 – this is *Conway's* edge of chaos. As a test, λ was plotted in a number of simulation runs with *K* = 2, and *R* ranging from 3 to 10. While *Conway* requires only *R* = 3, it was found that it easily evolves when *R* is greater than 3, even though some sub-rules become redundant or over-written. As expected, lambda converges to 0.273 (indicated by the arrow) as the dynamics approach *Conway*, as seen in the top graph in Figure 7, showing a simulation run with *R* = 10.

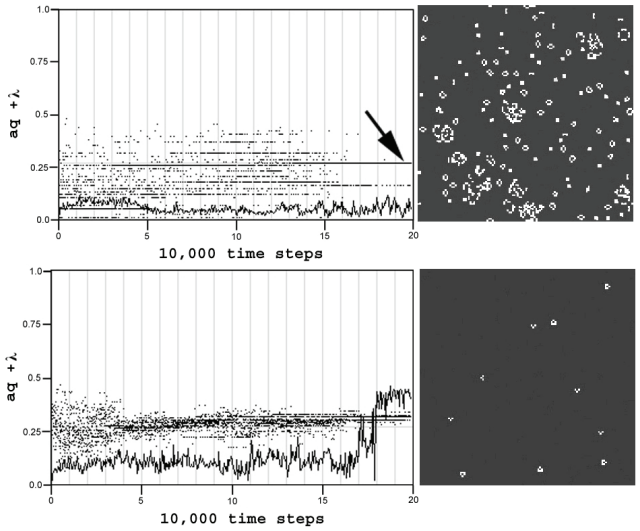

Figure 7: *Conway* (top) evolves with λ converging to 0.273. Another CA (bottom) with *K*=4 and *R*=8, also shows λ.

A few observations are worth mentioning. In simulation runs, *Conway* converges more slowly in general than CA with *K* > 2. As the graph shows, *aq* never gets very high. *Conway* is less glider-rich, and the gliders move more slowly than other CA. Thus, *Conway* offers less opportunity for particles to find a good ride, at least given the selection criterion used. The multiple horizontal lines in the λ plotting reveal the presence of competing regions in the CA domain, each gravitating toward different λ. At the end of the simulation run can be seen a band of lower λ values which competes with the λ for Conway, but begins to weaken at the very end.

The edge of chaos changes as *K* varies. The bottom of figure 7 shows the output of a simulation with *K*=4 and *R*=8. λ converges even though *aq* has not increased significantly indicating that genetic variety is decreasing. After around 160,000 time steps, *aq* increases, and λ converges more. In this particular CA, λ is slightly higher than 0.273.

Conclusions

Perhaps there is a fundamental digital nature to motion and the universe in general, as claimed by Edward Fredkin, (Wolfram, 2002), and others. But we find utility in modeling natural phenomena using higher-level, Newtonian models. Particle Swarms model motion (point-masses accelerated by velocities). And so they can be mapped to certain motion-like phenomena that emerge within CA dynamics. Setting particles to work over the space of CA can be a way of substituting our own smooth-tracking eye behavior, to detect the potential for glider

dynamics. But this is not the only reason for developing this technique. A swarm of particles can bring about emergent complexity (and perhaps universality) in CA by engaging in an interactive dance – it is a symbiotic relationship: it is *coadaptive*, and as such it models an important theory of life: that no life form emerges entirely on its own – there is always an environment, always other interacting agents involved.

The technique described in this paper applies some of the principles and techniques of Particle Swarm Optimization, as well as the basic operators of the Genetic Algorithm, to the study of emergent complexity in Cellular Automata. It is hoped that this hybrid technique will be a contribution to the study of complexity, and a tool to explore the nature of motion and emergent computation.

References

Berlekamp, E. R.; Conway, J. H.; and Guy, R. K. (1982). "What Is Life?" Ch. 25 in "Winning Ways for Your Mathematical Plays", Vol. 2. London: Academic Press.

Das, R., Mitchell, M. and Crutchfield, J. (1994). "A Genetic Algorithm Discovers Particle-Based Computation in Cellular Automata" In Parallel Problem Solving from Nature-III**,** Y. Davidor, H.-P. Schwefel, and R. Männer (eds.), Springer-Verlag 344-353

Gardner, M. (1970). Mathematical Games - The Fantastic Combinations of John Conway's New Solitaire Game "Life". Scientific American 223 (October 1970): 120-123.

Grassé, P.P. (1959). La reconstruction du nid et les coordinations interindividuelle chez Bellicositermes natalensis et Cubitermes. La theorie de la stigmergie: Essai d'interpretation des termites constructeurs. Insectes Sociaux, 6:41-83.

Kennedy, J. and Eberhart, C. (1995). Particle Swarm Optimization. Proceedings of the 1995 IEEE Conference on Neural Networks.

Langton, C. (1992). Life at the Edge of Chaos, Artificial Life II. Addison-Weskey. 41-91

Ramos, V., and Almeida, F. (2000). Artificial Ant Colonies in Digital Image Habitats - A Mass Behaviour Effect Study on Pattern Recognition, Proceedings of ANTS 2000. - 2nd International Workshop on Ant Algorithms. Marco Dorigo, Martin Middendorf & Thomas Stüzle (Eds.). 113-116

Reeves, W. (1983). Particle Systems – A Technique For Modeling A Class of Fuzzy Objects. ACM Transactions on Graphics Volume 2, Issue 2 (April 1983) 91-108.

Rendell, P. (2001). Turing Universality of the Game of Life. from Collision-Based Computing, ed. Adamatzky, A. Springer-Verlag. 513 – 539.

Shi, Y.H., and Eberhart, R.C. (1998). Parameter Selection in Particle Swarm Optimization. 7th Annual Conference on Evolutionary Computation, San Diego, USA. Springer.

Smith, A. R. III. (1971). Simple computation-universal cellular spaces. Journal of the Association for Computing Machinery, 18, 339–353.

Ventrella, J. (2000). Breeding Gliders with Cellular Automata. http://www.ventrella.com/Alife/Cells/cells.html

Wolfram. S. (1984). Universality and Complexity in Cellular Automata. Physica *D* 10: 1-35.

Wolfram. S. (2002). A New Kind of Science. Wolfram Media.

Wuensche, Andrew. (2001). Finding Gliders in Cellular Automata, from Collision-Based Computing, ed. Adamatzky, A. Springer-Verlag. 381-410