Interactive Evolution of Dynamical Systems

Karl Sims

1 ABSTRACT

Simple local rules for dynamical and cellular automata systems can give rise to relatively complex and interesting structures. This paper describes how rules for these systems can be bred to search for new and unusual examples of emergent behavior from dynamical systems. Random mutations and matings of rule sets, followed by user selection based on observations of resulting behaviors, allow a variety of different dynamical systems to be interactively evolved. Two examples of breeding rules for two dimensional dynamical systems have been implemented and are presented. The first involves cellular automata networks with rules represented by lookup tables. The second uses sets of variable length lisp expressions to describe the initial states, and differential equations for grids of state variables. Results suggest that these are powerful methods for creating dynamical systems with emergent complexity that would be difficult to build by design.

2 INTRODUCTION

Cellular automata (CA) networks can be useful tools for modeling complex dynamical systems including physical and biological systems. Repeatedly applied local rules which determine the new state of each cell from its current state and the states of its neighbors can give rise to surprising levels of complexity, physical accuracy, and even "life-like" behavior [3, 10, 13, 20, 21, 22, 26, 27]. Simulations involving locally applied rules can be preferable to those with globally operating rules because they are highly parallel in nature, and probably more analogous to real chemical and biological systems of millions of interacting molecules or cells [1, 14, 15, 23].

One of the challenges involved in creating and studying artificial life-like systems, is designing local rules which successfully give rise to interesting global behaviors. It can be difficult to specify and predict the effects of local rules on the overall system, especially as they become more complex. This paper proposes methods that allow new forms of local rules to be generated that result in interesting and complex dynamical systems, but the local rules are not required to be preconceived, designed, or even understood by the human creator.

In biological systems, DNA could be considered as the specification of local rules which are acted out by proteins. They, in turn, define the biochemical dynamical systems which result in the development and functioning of organisms. As it would currently be very difficult to design DNA sequences for new types of viable organisms, it may also be difficult for humans to specify the local rules for complex life-like simulations. Both natural evolution and the simulated evolution presented here involve the variation and selection of rules for dynamical systems. Although the dynamical systems created here are relatively simple, they demonstrate the ease of achieving emergent complexity by combining the techniques of locally controlled dynamical systems with the evolutionary process.

2.1 Simulated Evolution

Genetic algorithms are search techniques in which populations of test points are evolved by random variation and selection [5, 7]. They are employed in a number of applications to find optima in very large search spaces. Reproduction of *genotypes* with random variation, and selection of *phenotypes* based on a non-random *fitness* function drives a population of individuals towards higher and higher levels of fitness. Sexual reproduction allows desirable traits to evolve independently and later be combined into the same genotype.

The work presented here uses random variation of genotypes that represent rules for dynamical systems to allow searching spaces of possible dynamical systems. For each generation, the rules of each genotype are applied locally to the cells of 2D networks. Selection is performed on the phenotypes which are the resulting global behaviors of these systems. In this way, the direction of the simulated evolution is determined by these global behaviors, but mutations are performed on the genotypes representing the encoded local rules.

Population sizes used for genetic algorithms are typically fairly large (100 to 1000 or more) to allow searching of many test points and avoiding local optima. For interactive efficiency and user interface practicality the examples presented here use a much smaller population size (4 - 16) and only one or two individuals are chosen to reproduce for each new generation.

2.2 Interactive Selection

In the work described here, the fitness of dynamical systems is determined interactively by a user at each step of the evolution

process instead of automatically by a pre-defined fitness function. *Perceptual selection* is used because fitness functions that could determine how interesting or aesthetically pleasing a dynamical system is would be difficult to define, and many local optima may be of interest instead of just one global optimum. This allows the user to not only observe the intermediate results as the evolution progresses, but also to interactively navigate through the spaces of possible results.

In *The Blind Watchmaker*, Dawkins demonstrates the power of Darwinism with a simulated evolution of 2D branching structures called "biomorphs." Here, the user also interactively selects the shapes that survive and are reproduced to create each new generation [4].

2.3 Overview of the Paper

Two different methods for representing rules for 2D dynamical systems have been implemented. The first requires the value of each cell to be an integer of fixed length. A lookup table determines the next state of each cell from an index made from its current state and those of its neighbors. This allows all possible CA rules for a given number of bits per cell to be described by filling the lookup table with different values. In this case, the lookup table is the genotype which can be mutated and mated with other lookup tables to evolve CAs with various behaviors.

The second representation for genotypes of rules for dynamical systems contains lisp expressions that determine the initial state and the change of state of the system over time. Each cell can contain one or more "continuously" varying values (e.g. floating point numbers) instead of fixed-length integers. The initial state is determined by lisp expressions that return values for each cell location. Non-linear differential equations describe the behavior of the system, and are also represented by lisp expressions that calculate time derivatives for each cell as functions of the current state and neighboring states.

In the next section, the breeding of lookup tables for CA systems will be described. In section 4, breeding continuous dynamical systems with genotypes containing lisp expressions will be presented. Finally in sections 5 and 6, results are given and future work is suggested.

3 BREEDING CA LOOKUP TABLES

CA networks that contain a limited number of bits per cell can be represented and quickly simulated by using lookup tables to find the new state of each cell from an index of its current state and the state of its neighbors. In this work, CA lookup tables are interactively evolved by the following process:

1. A population of initial random lookup tables are generated by randomizing their bits. If zeros and ones are created with equal probability, chaotic behavior usually results, so zeros are created with a lower probability such that structured behavior is more likely (around $1/8$ zeros is used).

2. For each lookup table, a 2D grid of cells is created and the contents are initialized to random values. The values are updated using the table for a number of iterations (30 - 200). Each iteration is displayed to the user in real-time as the simulation proceeds, by mapping the cell values into pixel intensities and displaying the grids of cells as images.

3. The user observes the animated behavior of the CA systems for each of the lookup tables, and selects one or more to survive.

4. The lookup tables corresponding to the selected systems are reproduced with mutations or combined with each other to create new lookup tables for the next generation.

This process of perceptual selection of CA behaviors, and reproduction with variation of the lookup tables is repeated (steps 2 - 4). Figures 1 - 3 show a variety of results that can occur after a number of generations. (Typically around 5 - 20 are required.)

3.1 Mutating and Mating CA Tables

For new variations of CA networks to occur, the lookup tables must be reproduced with some frequency of mutations, as stated in step 4 above. The method used here involves subjecting each bit in the table to probabilistic inversion. (An inversion frequency of around 0.01 is used because it causes frequent variation in results, but still provides some stability.) A non-local approximation of this method can require generation of fewer random numbers: the locations are chosen at random for a constant number of mutations.

Sexual combination of two parent lookup tables is performed by *crossing over* information between the parents to generate a new table. Values are copied from one of the parent tables, but with some frequency, the source table is switched to the other parent. This causes connected segments of the table to be more likely to stick together then sections at distant locations. (A frequency of .001 is used here such that only a few crossovers are likely in any one mating.) Again, an approximation to this method can save random number generation by choosing random locations for a constant number of crossovers.

3.2 Limitations of Representation

Although CA lookup tables can produce interesting behaviors, several limitations are noticeable: tables can become very large when the length of the state integers is more than a few bits; there is a limited number of states that each cell can have; and the space of possible tables is highly dimensional but still limited in its extent.

If the state of each cell contains N bits, and 3x3 neighboring states are used to determine each new state, the table will contain new states for each of 2^{9N} possibilities and is $N2^{9N}$ bits in length. This is acceptable for states of 1 bit where the table length is 512 bits, but for 2 bit states the table is already 524,288 bits long, which can prevent mutations and matings from being calculated quickly. Several modifications can help reduce the size of lookup tables:

1. The number of bits taken from the neighbors can be reduced to a subset of the total bits of their states.

2. The bits of neighbors can be combined with various associative operators and the results used in the table index instead of the neighbor bits themselves. (For example, and, or, xor, min, max, or $+$, might be used.) This limits the CAs to symmetrical behav-

ior, but this can actually be advantageous since asymmetrical rules commonly give undesirable directional shifting of the system.

3. Another useful method for shortening lookup table sizes involves shifting and wrapping the table on top of itself. Each bit in the table can be a part of more than one new state values. For example, if states consist of 4 bits, the lookup table can be shifted by 0,1,2, or 3 bits to provide 4 times the number of new states with the same table length. Tables could be read backwards and scrambled in various ways to provide even more shortening if necessary. The consequences of duplicating the effects of the lookup table bits have not been evaluated in detail, but it has been experimentally observed to be effective. Some viruses have developed a strategy similar to this – certain regions of RNA can be read twice, once shifted over by a base pair, to encode two different proteins [2].

These methods have allowed lookup tables for CA networks with 3, 4, or more bits per state to be mutated, mated, and simulated at interactive rates. [Figures 1 - 3].

A second limitation of using lookup tables to describe dynamical systems is that they only allow the states to have a fixed number of integral values. This tends to give the system a quantized look with regular shapes and blocks of pixels. Instead, it might be desirable to have cells contain values that can vary continuously.

A final limitation of evolving CA lookup tables, is that the extent of possible results is fixed. Although the dimensionality is very high, all genotypes are essentially equally complex and can not evolve towards higher levels of complexity. The parameters used to express the genotype are fixed; the number of bits in the state of each cell and the operations for combination of neighbors can not be modified or extended by mutations.

4 BREEDING CONTINUOUS DYNAMICAL SYSTEMS

In an attempt to surpass the limitations described above, a second representation for dynamical systems is presented. Arbitrary differential equations for continuously varying state variables are described by hierarchical lisp expressions which can be mutated, mated with each other, and evaluated to perform simulations.

4.1 Lisp Expressions as Genotypes

Traditionally, genotypes consist of fixed-length strings of digits or parameters, such as the ones described above. Fixed-length genotypes and fixed expression rules limit the phenotypes to that pre-defined space of possible results. Koza has used lisp expressions as genotypes such that the dimensionality of the search space itself can be extended to solve problems such as artificial ant navigation and game strategies [8, 9]. Discovery systems, such as AM and Eurisko, also utilize a form of mutating lisp programs [11]. Recently, artificial evolution of lisp expressions has been used to generate unusual pictures and textures for computer graphics [17].

In this application, lisp expressions are used as genotypes to determine the initial states and time derivatives for variables of continuous dynamical systems. For example, a system containing two quantities, A and B, at each grid location is described by four equations:

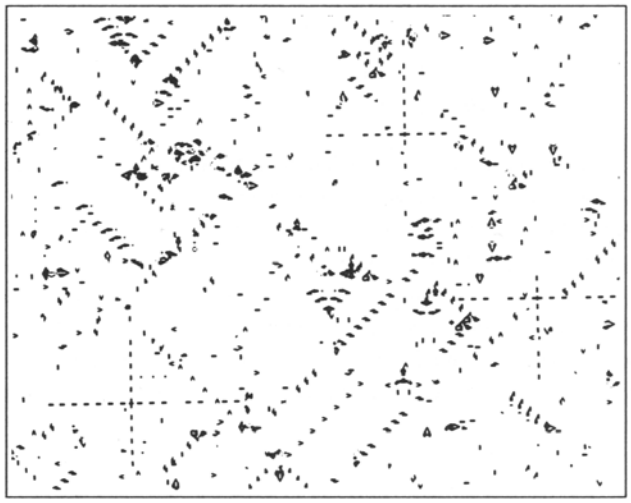

Figure 1: Evolved CA – Glider Zoo

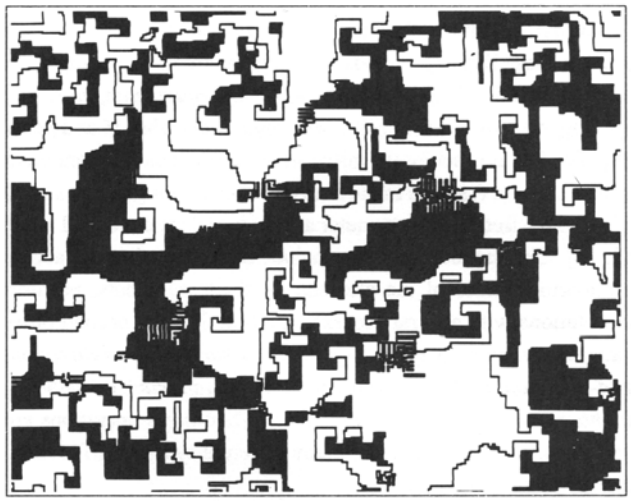

Figure 2: Evolved CA – Spiraling Regions

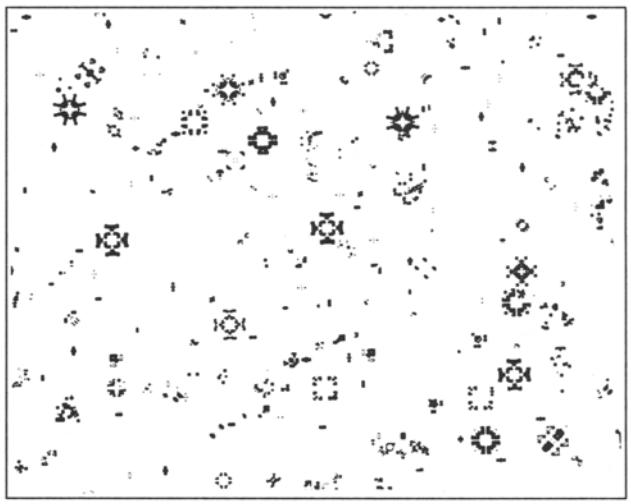

Figure 3: Evolved CA – Sparkles

$$A_0 = F_{A0}(X, Y)$$
$$B_0 = F_{B0}(X, Y)$$
$$dA/dt = F_{dA}(A, B)$$
$$dB/dt = F_{dB}(A, B)$$

F_{A0} and F_{B0} are functions that determine the initial values for each element of A and B from their grid locations (X, Y). F_{dA} and F_{dB} are functions that determine the time derivatives for each element of A and B using the current state of the system. Arbitrary functions for F_{A0}, F_{B0}, F_{dA}, and F_{dB}, are specified by lisp expressions which can vary in size, structure, and behavior. A genotype of lisp expressions that would describe a simple reaction-diffusion-like system of two chemicals that diffuse and inhibit each other might be:

A_0 = (noise .8 .4)
B_0 = (noise .9 .5)
dA/dt = (− (laplacian A) B)
dB/dt = (− (laplacian B) A)

The list of primitive functions, or *function set*, that can be chosen to create these lisp expressions, contains operators including standard common lisp functions: +, −, ∗, /, *mod, round, min, max, abs, expt, log, sin, cos, atan, negate, sqrt, square, dissolve, if,* and *plusp* [18]. The function set for the initial state expressions also contains a *noise* procedure (as used in the example above) that generates solid noise from a frequency parameter and an initial random-seed value [12, 16]. The function sets also include operations that can access neighboring values of the elements of their arguments, perform convolutions with arbitrary masks, and find first and second order spatial derivatives: *x-grad, y-grad, grad-mag, grad-in-direction, grad-direction, laplacian, anisotropic-laplacian, curl, convolve-5-neighbors,* and *convolve-9-neighbors*. When these operations are used in various combinations, many different types of differential equations can be specified.

Simple random expressions are created for the initial state and time derivatives of each state variable. A random expression is generated by choosing from: a random constant value, an input variable (such as X, Y, A, or B), or a function from the function set with recursively generated random expressions for arguments. Interactive evolution is performed by first creating several genotypes with expressions generated in this way, and then displaying the corresponding simulations to the user. The state variables are mapped into colors for each iteration to visualize the animated behavior of the system in real-time. Then, the user selects one or more of these systems for mutation and mating to produce the next generation, and the process repeats. After a number of generations, genotypes with fairly complex expressions and interesting resulting behaviors can occur. As an alternative to starting with randomly generated expressions, the user can hand code an initial set of equations, such as a wave equation or a reaction-diffusion system, and begin the evolution from there. This allows many variations of input systems to be explored.

4.2 Mutating and Mating Lisp Expressions

A recursive mutation scheme is used to mutate genotypes containing lisp expressions. Each expression is traversed as a hierarchical structure and each node is in turn subject to possible mutations. Each type of mutation occurs at different frequencies depending on the type of node:

1. Any node can mutate into a completely new random expression.
2. If the node is a scalar value, it may be adjusted by the addition of some random amount.
3. If the node is a function, it can mutate into a different function. For example *(abs A)* might become *(cos A)*. If this mutation occurs, the arguments of the function are also adjusted if necessary to the correct number and types.
4. An expression can become the argument to a new random function. If necessary, other arguments are generated at random. For example, A might become *(∗ A .3)*.
5. Finally, an argument to a function can jump out and become the new value for that node. For example *(∗ A .3)* might become A. This is the inverse of the previous type of mutation.

Other types of mutations could certainly be implemented, but these are sufficient for a reasonable balance of slight modifications and potential for changes in complexity. The overall mutation frequency is scaled inversely in proportion to the length of the entire expression. This decreases the probability of mutation at each node when the parent expression is large so that some stability of the phenotypes is maintained. To keep evaluation of these expressions at real-time speeds, estimates of computation times are made, and slow expressions are automatically eliminated before being used.

Lisp expressions can be mated by *crossing over* sub-expressions between two parent expressions. A node in the expression tree of one parent is chosen at random and replaced by a node chosen at random from the other parent. This allows two sub-expressions that have evolved independently to be combined into a single genotype. Two genotypes are mated by mating each pair of expressions that specify each initial state and each time derivative.

4.3 Dynamic Simulation

For simplicity, simulations of continuous dynamical systems are performed using Euler's method of integration. The differential equations are approximated for a small discrete time interval $\triangle t$. For example,

$$\frac{dA}{dt} = F(A)$$

would be simulated by computing many discrete updates of the value of A:

$$A' = A + \triangle t F(A)$$

When $\triangle t$ is smaller, the simulation is more accurate, but more computation is required. ($\triangle t = 0.1$ is often used.)

Systems can sometimes generate values that exceed the legal bounds of numerical representation. Values are regularly clamped

to some legal bounds to avoid overflow errors. These particular discretizations of time and clamping parameters can affect the behaviors of some systems. In fact, systems sometimes evolve that exploit these specific procedures for interesting effects.

Higher order differential equations can also evolve. For example, a simple wave propagating system that indirectly specifies acceleration (d^2B/dt^2), instead of just velocity (dB/dt), might be described by:

$A_0 = .0$
$B_0 = (\text{noise } .1\ .4)$
$dA/dt = (\text{laplacian B})$
$dB/dt = \text{A}$

The first derivatives are also included as possible arguments in the expressions to allow for further possibilities. Resulting behaviors might not be consistent if $\triangle t$ is modified, but for a given time increment, this can help interesting physical-like systems to occur. For example, the previous system might instead be described with one state variable:

$A_0 = (\text{noise } .1\ .4)$
$dA/dt = (+\ dA/dt\ (*\ .1\ (\text{laplacian A})))$

Similarly, expressions can evolve that counteract the incremental integration, and specify the next state directly from the current state.

The space of possible dynamical systems can be further enhanced by allowing complex numbers, instead of just real values, to be included in the state variables and expressions. The operations in the function set are adjusted to perform on complex quantities as well as reals, and complex constants and a grid coordinate value, $\#C(X\ Y)$, are included as possible arguments. (The form $\#C(r\ i)$ is used to denote a complex quantity with real part r and imaginary part i.) Expressions of real scalar values could theoretically evolve to perform the same complex operations, but this might not be likely to occur in a practical timeframe. Various spiral shapes and fractal structures have been found that use complex arithmetic [Figures 8 - 11].

5 RESULTS

These two techniques for interactive evolution of dynamical systems have been implemented on the Connection Machine(R) system CM-2, a data-parallel supercomputer [6, 19]. One virtual processor is assigned to each cell in the network so the entire grid can be processed simultaneously. Identical copies of lookup tables are distributed into every group of 32 processors so all processors can perform parallel table lookups using local memory references. Lisp expression mutations and crossovers are performed on a *front-end* computer, and the expressions are evaluated in parallel using the Connection Machine system and *Starlisp*. These implementations allow grid dimensions of up to 256x256 to be simulated at interactive rates, depending on the complexity of the genotype. Dimensions of 512x512 and larger can result in less efficient interactivity, but are still useful for viewing systems with high levels of detail.

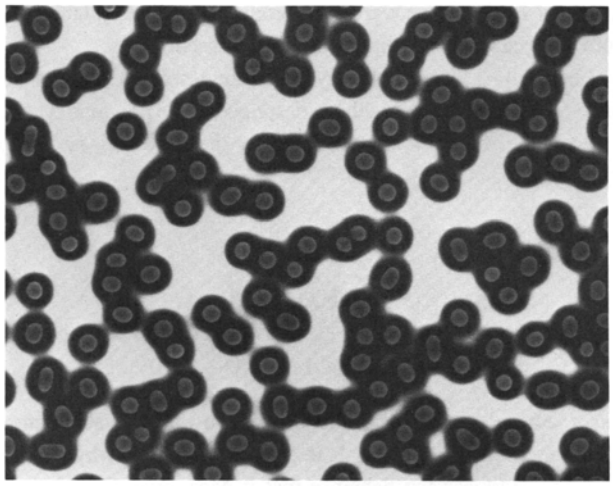

Figure 4: Cell Shapes

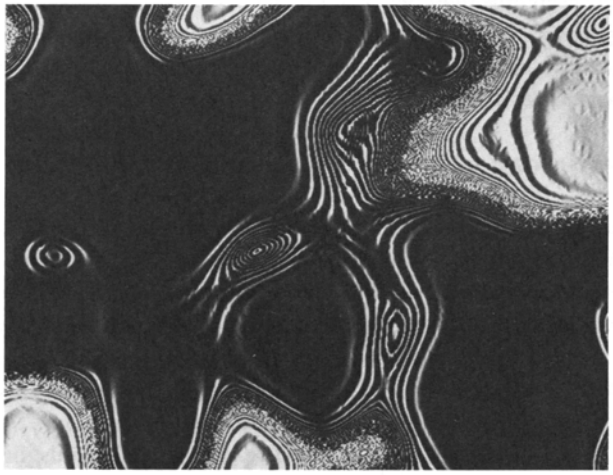

Figure 5: Wave Generators

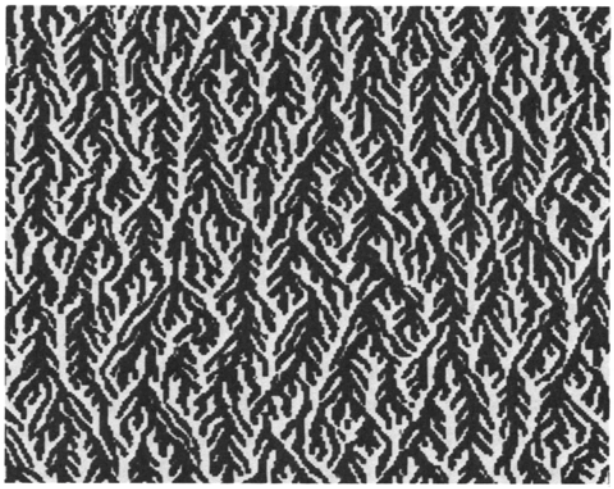

Figure 6: Branching Patterns

Examples of some different types of CA behaviors are shown in figures 1 - 3. These were each produced by lookup tables that were evolved by the methods described in section 3.

Figures 4 - 13 show a variety of continuous dynamical systems that were evolved by the methods described in the previous section. For example, figure 4 was produced by the following system of equations:

```
A0 = (sin (noise -.14 -.77))
B0 = 1.99
dA/dt = (+ (+ (laplacian A 2.1)
           (if-plusp (− A B) .4 .0)) (∗ -.38 A))
dB/dt = (+ (laplacian A 4.99) (∗ -.4 B))
```

This system proceeds from random noise towards a stable pattern of circular cell-like shapes. It is often not obvious why a set of equations produces the behavior that it does, even for relatively short expressions. Fortunately, a complete understanding of these equations is not required even by the creator. The expressions that specify the rules that produced figures 5 - 13 are given in the Appendix. Genotypes such as these can be interactively evolved in timescales such as 10 minutes – probably much faster than they could be designed.

6 FUTURE WORK

Many extensions to these techniques could be explored. Networks with connectivity other than 2D rectilinear grids might be represented and evolved. The number of elements and connections themselves could be subject to mutation and evolution. Dynamical systems similar to these but in three dimensions, could be simulated, visualized, and evolved, although the size of the volumes that could be processed in near real-time would have severe limits.

Images from other sources could be incorporated into the lisp expressions for initial states and differential equations of dynamical systems. This would allow arbitrary input images to determine the initial states or various dynamical properties of evolved systems, and might result in some unusual effects.

Fitness functions other than the interactive perceptual method could be used to direct the evolution of dynamical systems automatically. Algorithms which try to detect "interesting" behavior of moving images could be tested by observing the results of simulated evolutions which use those algorithms as fitness functions. Perhaps the information from many human selection decisions could be generalized and used to help define an automatic fitness function.

7 CONCLUSION

The work presented here attempts to combine the benefits of several techniques: locally specified rules for dynamical systems, evolution by random variation and non-random selection, and genotypic representations of variable complexity. It is likely that systems of evolving biological life have also utilized combinations of these techniques.

Interactive evolution is a potentially powerful method for creating and exploring complexity that does not require human understanding of the specific process involved. It could be considered a tool for helping a user with creative explorations, or it might be considered a system which attempts to "learn" about what is interesting from a human. In either case, it allows the user and computer to work together to construct results that neither could easily produce alone.

Interactive evolution of many types of dynamical systems should become more practical as computation becomes more powerful and available, and the techniques presented here will hopefully contribute to creating systems that give rise to emergent behaviors of higher and higher levels of complexity.

8 Acknowledgments

Thanks to Lew Tucker, Gary Oberbrunner, Matt Fitzgibbon, Jim Salem, and Peter Schröder for help and CM graphics software support. Thanks to Richard Dawkins for the interactive concept. Thanks to Pattie Maes for encouragement, thanks to Katy Smith for proofreading, and thanks to Bruce Boghosian for differential equation tips.

9 APPENDIX

Figure 5, Wave Generators:

```
A0 = 0.33
B0 = 0.27
C0 = (log (− 0.5 (grad-mag-squared (noise -0.2 -0.04))) (/ (noise 0.02 0.03) (noise
-0.007 -1.4)))
dA/dt = C
dB/dt = (anisotropic-laplacian (sin A) A 0.9 0.08)
dC/dt = (neighbor-ave (atan dA/dt (laplacian B 1.8)))
```

Figure 6, Branching Structures:

```
A0 = Y
B0 = 1.0
C0 = (+ (negate (noise 0.12 1.9)) Y)
dA/dt = (neighbor-max (neighbor-max C))
dB/dt = (x-grad C)
dC/dt = (neighbor-ave (grad-direction B 0.25))
```

Figure 7, Crack Patterns:

```
A0 = (− (noise -0.064 1.17) -1.58)
B0 = -0.032
dA/dt = (neighbor-min A)
dB/dt = (laplacian A 4.99)
```

Figure 8, Globe:

```
A0 = #C(X Y)
dA/dt = (+ (/ dA/dt A) (+ (grad-direction (expt #C(1.6 0.25) 3.5) -0.42) (x-grad A)))
```

Figure 9, Fractal Spirals and Arms:

```
A0 = #C(X Y)
dA/dt = (+ (/ (+ (square A) 1.0) A) (+ -0.7 (expt (max (max A (laplacian (log A #C(-1.2
-0.05)) 0.11)) #C(0.21 -0.12)) 3.5)))
```

Figure 10, Spiral Wave:

```
A0 = #C(X Y)
dA/dt = (+ (/ (min A 1.0) A) (+ -0.7 (expt (max A #C(0.2 -0.12)) 3.5)))
```

Figure 11, Growing Fractal Buds:

```
A0 = #C(X Y)
dA/dt = (+ (/ (+ (square dA/dt) 1.05) A) (+ -0.7 (expt (max A #C(0.21 -0.12)) (max A
#C(0.21 -0.12)))))
```

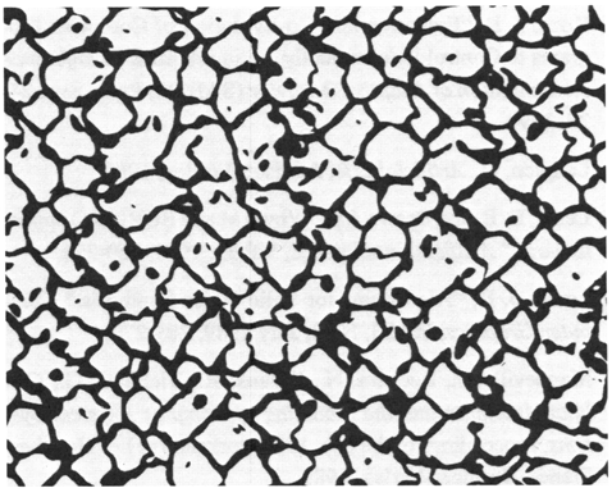

Figure 7

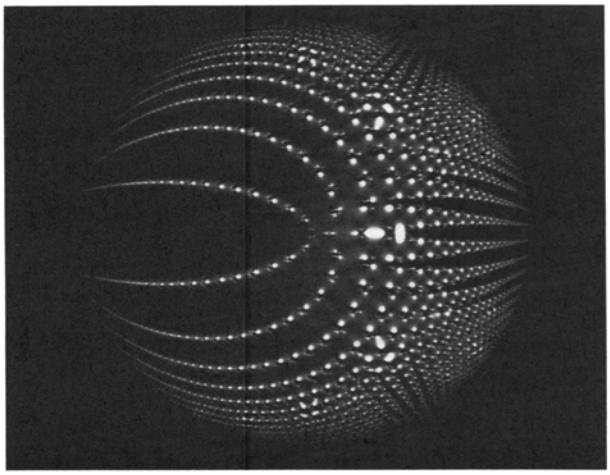

Figure 8

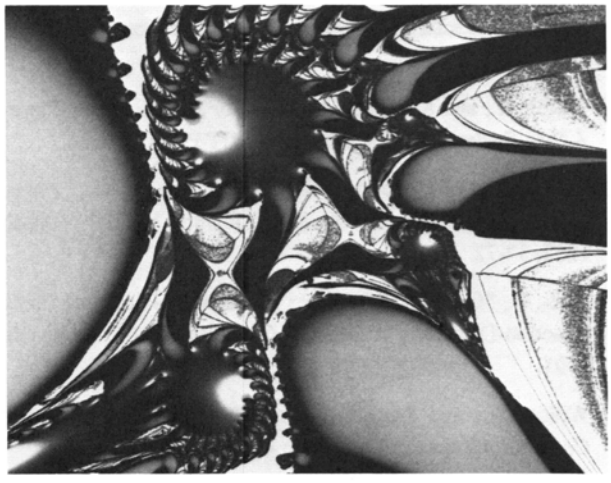

Figure 9

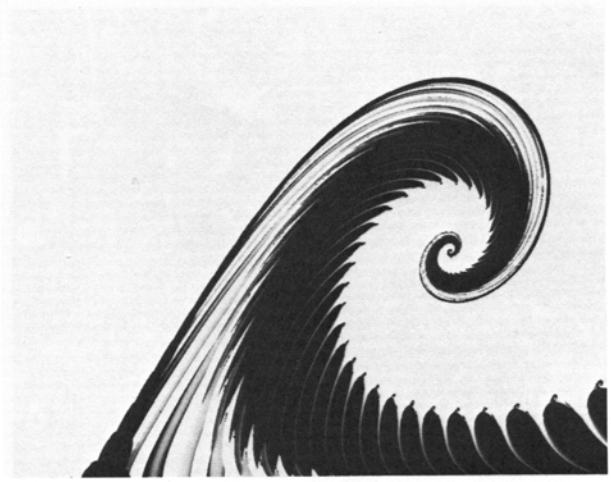

Figure 10

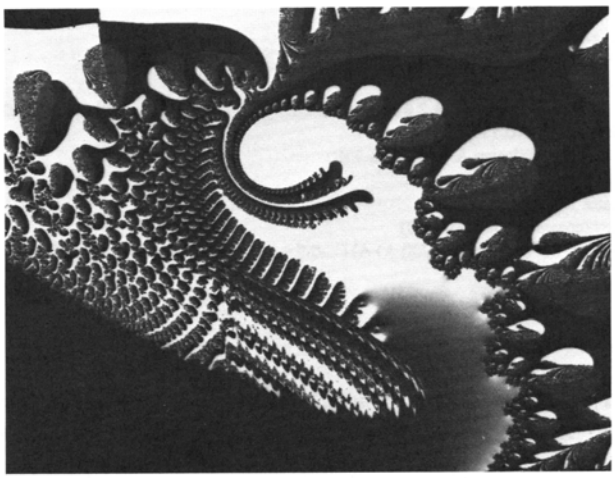

Figure 11

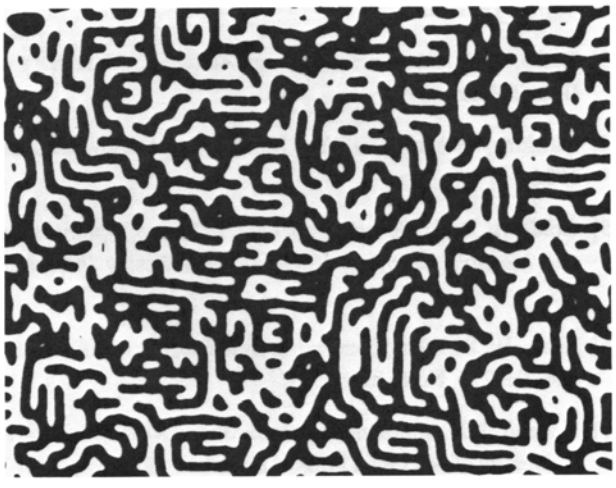

Figure 12

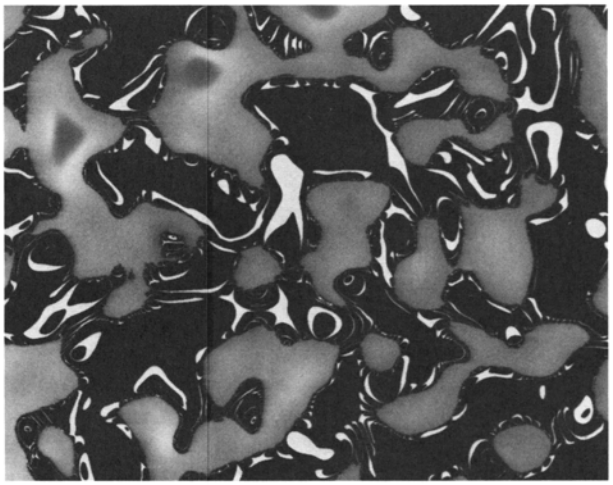

Figure 13

Figure 12, Reaction Diffusion Pattern:
A0 = (negate (noise -0.14 -0.77))
B0 = -0.086
dA/dt = (+ (+ (laplacian A 2.0) (convolve-5-neighbors-constant (- A B) 0.4 0.0 0.027 0.27 0.66)) (* -0.4 A))
dB/dt = (+ (laplacian A 4.99) (* -0.4 B))

Figure 13, Striped Blobs:
A0 = (complex-noise 0.06 1.3)
dA/dt = (+ (/ (+ (square dA/dt) A) A) (+ -0.7 (expt (max A #C(0.2 -0.12)) 2.8)))

References

[1] Babloyantz, A., *Molecules, Dynamics, and Life: An Introduction to the Self-Organization of matter*, Wiley Interscience, New York, 1986.

[2] Beremand, M. N., and Blumenthal, T., "Overlapping Genes in RNA Phage: A new Protein Implicated in Lysis," *Cell*, Vol.18, 1979, 257-266.

[3] Burks, A.W., *Essays on Cellular Automata*, University of Illinois Press, 1970.

[4] Dawkins, Richard, *The Blind Watchmaker*, Harlow Logman, 1986.

[5] Goldberg, D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, 1989, Addison-Wesley Publishing Co.

[6] Hillis, W. D., "The Connection Machine," *Scientific American*, Vol.255, No.6, June 1987.

[7] Holland, J. H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

[8] Koza, J. R. "Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems," Stanford University Computer Science Department Technical Report STAN-CS-90-1314, June 1990.

[9] Koza, J. R. "Evolution and Co-Evolution of Computer Programs to Control Independently Acting Agents," *Conference on Simulation of Adaptive Behavior* (SAB-90) Paris, Sept.24-28, 1990.

[10] Langton, C., *Artificial Life*, Addison-Wesley, 1989.

[11] Lenat, D. B. and Brown,J.S. "Why AM and EURISKO appear to work," *Artificial intelligence*, Vol.23, 1984, 269-294.

[12] Lewis, J. P., "Algorithms for Solid Noise Synthesis," *Computer Graphics*, Vol.23, No.3, July 1989, 263-270.

[13] Manneville, P., Boccara, N., Bidaux, R., Vichniac, G., *Cellular Automata and the Modeling of Complex Physical Systems*, Proceedings of the Feb. 1989 workshop at Les Houches, France, Springer-Verlag, 1989.

[14] Meinhardt, H., *Models of Biological Pattern Formation*, Academic Press, London, 1982.

[15] Murry, J., "How the Lepard Gets its Spots (biological inquiry into single pattern-formation mechanism in animal coats)" *Scientific American*, Vol.258, 1988, p.80.

[16] Perlin, K., "An Image Synthesizer," *Computer Graphics*, Vol.19, No.3, July 1985, 287-296.

[17] Sims, K., "Artificial Evolution for Computer Graphics," *Computer Graphics*, Vol.25, No.4, July 1991.

[18] Steele, G., *Common Lisp, The Language*, Digital Press, 1984.

[19] Thinking Machines Corporation, *Connection Machine Model CM-2 Technical Summary*, technical report, May 1989.

[20] Tamayo, P., and Hartman, H., "Cellular Automata, Reaction-Diffusion Systems, and the Origin of Life," *Artificial Life*, Addison-Wesley, 1989, 105-124.

[21] Toffoli, T., "Cellular Automata as an alternative to (rather than an approximation of) differential equations in modeling physics," *Physica* 10, North-Holland, Amsterdam, 1984, 117-127.

[22] Toffoli, T., and Margolus, N., *Cellular Automata Machines: A New Environment for Modeling*, MIT Press, 1987.

[23] Turing, A., "The Chemical Basis of Morphogenesis," *Philosophical Transaction of the Royal Society*, Vol.237, August 1952, 37-72

[24] Turk, G., "Generating Textures for Arbitrary Surfaces Using Reaction-Diffusion," *Computer Graphics*, Vol.25, No.4, July 1991.

[25] Witkin, A., and Kass, M., "Reaction Diffusion Textures" *Computer Graphics*, Vol.25, No.4, July 1991.

[26] Wolfram, S., "Cellular Automata as Models of Complexity," *Nature*, Vol.311, 1984, p.419.

[27] Wolfram, S., *Theory and Applications of Cellular Automata*, World Scientific, 1986.