Complex Systems **3** (1989) 79–90

Symbiotic Programming: Crossbreeding Cellular Automaton Rules on the CAM-6

Rudy Rucker
Department of Mathematics and Computer Science,
San Jose State University, San Jose, CA 95192, USA

1. Introduction

One way to evolve new artificial lifeforms is to fuse existing forms in various ways. If one wishes for a new lifeform's program to be the same size as each of its parent programs, then one might take half of each parent program and patch the two halves together. This method is typical of sexual reproduction, and of the genetic programming approach.

A different method of crossbreeding two of three lifeforms is to write a double or triple length program which includes the full code of the original programs and which provides an arena in which the programs can interact. This might be called a symbiotic programming approach.

In this paper I will describe some new and lively cellular automaton rules which I have found through symbiotic programming on the CAM-6 cellular automaton machine [1,4].

2. The CAM-6

The CAM-6 cellular automaton machine consists of a PC board and a Forth software environment. The board and software were developed by Tommaso Toffoli and Norman Margolus, both at the Massachusetts Institute of Technology Laboratory of Computer Science. The CAM-6 board with software is in principle available for $1500 from Systems Concepts of San Francisco; unfortunately, Systems Concepts' filling of orders to date has been extremely slow.

The CAM-6 treats a color computer monitor as a 256 by 256 array of cells. Each cell holds four bits of information. Often one thinks of the screen as being a stack of four bitplanes which are displayed as different pixel colorings.

The cells are updated in parallel. Each cell checks itself and its neighbors, and then calculates its next state on the basis of a shared lookup table. In most of the rules I have looked at, a cell is able to "see" a total of twelve neighbors: the nine members of its 3×3 neighborhood in its own bitplane,

also the three cells directly above or below it in the other three bitplanes. Usually a cell is called Center and its planar neighbors are called North, Northeast, East, Southeast, South, Southwest, West, and Northwest. One way of referring to adjacent cells in the other planes is to speak of Center0, Center1, Center2, and Center3. If Center is in Plane 0, then it is the same as Center0.

The most remarkable feature of the CAM-6 is it speed. It updates the whole screenful of cells 60 times a second. At 4 bits per cell, this means the CAM-6 is generating $256 \times 256 \times 4 \times 60 \simeq 16$ million display bits per second.

Another remarkable feature of the system is the shortness of the programs which produce interesting results. This is in fact a property of cellular automata in general: a well chosen cellular automaton rule of only a few dozen bits can quickly convert a start screen with only a dozen bits lit into a full 256K bit color screen seething with activity.

Some screens die or enter tight loops, other boil randomly, and at the interface are the artificial life screens which show pseudopurposeful activity. Wolfram [7] classifies these rules as, respectively: class 1&2, class 3, and class 4; Langton [3] calls them, respectively: quiescent, chaotic, and balanced. These distinctions are not absolute, as many interesting balanced rules will eventually move off the interface to die out or to go fully chaotic. In practice I have been interested in CAM-6 rules which can maintain balanced behavior for at least five thousand steps.

3. The three parent rules

The crossbreeding experiments I will describe here use the following three rules: John H. Conway's Life, Gerard Vichniac's Vote, and Brian Silverman's Brain [4,6].

In each of these rules we regard a cell as holding either a zero or a one. A cell's next state is determined by the states of the members of the cell's 3×3 neighborhood. The sum of all the cell values in this neighborhood is called the 9Sum, the sum of all the cell values save the central one is called the 8Sum.

The rule for Life can be succinctly represented in pidgin Forth as:

8Sum { 0 0 Center 1 0 0 0 0 0 0 } → Plane0

This notation means that for a given "Center" cell we form the 8Sum of its planar neighbors, and then use this total to select our cell's next state from a list of nine possibilities — corresponding to possible 8Sums of zero through eight. If a cell has two living neighbors, then its new value is the same "Center" value as before. If a cell has three living neighbors its value changes to 1. Otherwise its value goes to 0. These values are all displayed in Plane0.

If we run Vote in Plane1, its rule takes the form:

9Sum { 0 0 0 0 1 0 1 1 1 1 } → Plane1

This means that each cell's new state is determined by taking a vote among the nine members of its 3×3 planar neighborhood. But instead of determining the outcome by simple majority, near ties are awarded to the loser!

The rule for Brain requires the use of two planes; suppose we think of Brain as being a combination of two rules Brain2 and Brain3, running in planes 2 and 3.

$$[(8\text{Sum} = 2)\text{AND}(\text{Center2} = 0)\text{AND}(\text{Center3} = 0) \rightarrow \text{Plane2}$$
$$\text{Center2} \rightarrow \text{Plane3}$$

The understanding in the rule for Plane2 is that the compound sentence in square brackets has a Boolean truth value of 1 or 0. "Center2" stands for the Plane2 cell being updated, and "Center3" stands for the cell directly over it in Plane3. At each step, the present value Center2 is echoed in Plane3 as Center3. A cell is changed to 1 in Plane2 only if three conditions hold: the cell has two "live" neighbors, the cell is now off, and the cell was off during the last cycle as well.

Each of these three rules is very interesting to look at by itself. If we start any of these rules on a plane filled with a random soup of half 1s and half 0s, the following behaviors occur.

Life boils chaotically for a few cycles, then damps down leaving irregular active patches crawling about. The background at this stage consists of small static patterns (blocks and beehives) and isolated oscillators (blinkers). Any added cell, e.g. a wayward glider, can touch of an active patch which will move about for a few hundred cycles. But in all the pure Life runs I have observed, the screen soon dies down to hold nothing but small scattered periodic patterns.

Vote quickly forms small connected domains of 0s and 1s. A simple majority vote (given by 9Sum { 0 0 0 0 0 1 1 1 1 1 }) will freeze up at this stage and exhibit no further changes (though there can be small oscillators on the boundaries). Vichniac's vote keeps evolving, with more and more patterns fusing, till one normally has one or two large domains each of 0s and 1s. The CAM-6 screen is treated as a torus, so a patch which runs off the screen's left is contiguous with a patch running off the right. Typically, Vote will go to either all 1s or all 0s with a few small oscillator islands of the opposite parity; occasionally it will go to a pattern consisting of two horizontal stripes or two vertical stripes of opposite parity. The stripes' boundaries display long-cycle periodic behavior.

Thus, from a random start, Life and Vote exhibit "balanced," "artificially alive," or "class 4" behavior, but only for a short time. Their configuration space is densely seeded with point attractors and small-cycle attractors. Brain, on the other hand, seems to have a single large strange attractor in its configuration space. From practically any random start at all, Brain evolves to a very active balanced pattern filled with large amounts of structure.

The most common Brain patterns are "haulers," which move in each of the four principal grid directions. A hauler consists of two firing blocks followed

by two refractory blocks. Chains of outriggers attach themselves to haulers and are towed along. Sparks jump up and down the outrigger chains, spewing out writhing clots of debris. Just as Life has its special glider pattern, Brain's special pattern is the "butterfly." Like the glider, the butterfly is a small cell structure which propagates along the grid's diagonals at "one fourth the speed of light" (one cell for every four updates). I discovered a way to build a butterfly gun in June 1988.

Though Brain is not chaotic, it is hyperactive, with all known patterns but one being in constant motion. In January 1989 I found a static pattern, the "twizzler." The twizzler can be started as a block of four firing cells with four refractory cells attached, one to each side in a pinwheel symmetry.

In studying Brain, I developed several tricks to slow it down. One trick is to ignore one neighbor (say East) when forming the 8Sum. Another trick is to add lethal barrier lines to prevent the pattern from wrapping around the screen. A third way of slowing Brain down is to make the cells have two or even three clockcycles of being "refractory" after firing.

If one starts with a very simple pattern — as opposed to a random screen of half 0s and half 1s — Life can sometimes generate a lot of structure before going periodic, Vote will die boringly, and Brain will usually end up at the same strange attractor that it approaches from a random start.

4. New rules

I will describe three new rules: Brainlife, Votelife, and Ranch. For the sake of comprehensibility, I will not describe the programs in pure Forth, but will instead use a kind of mixture of Forth, natural language, and Pascal.

In Brainlife, I run Life in Plane0, and I run Brain in planes 2 and 3. We let the two rules interact only at locations where the Plane1 cell holds a 1. Plane1 is used to hold a static mask which allows the two rules to interact.

[(Life =1) OR ((Center1 =1) AND (Center2 =1))] → Plane0
Center1 → Plane1
[(Brain2 =1) AND NOT ((Center1 =1) AND (Center0 =1))] → Plane2
Center2 → Plane3

The interaction between Life and Brain is such that the presence of a Brain cell in the Plane1 mask region turns Life cells on. This keeps Life from dying out. The presence of a Life cell in the Plane1 mask region turns off any live Brain cell which touches it. This has the effect of damping Brain's activity.

In Plates 1 and 2 I show Brainlife evolving from a start consisting of a disk in plane 1 and 2 bits in Plane2. Brain evolves two adjacent bits into an expanding diamond pattern of bits in Plane2 and Plane3. We see the pattern crossing the mask in Plate1, and Plate2 shows the sequel.

In Votelife, I use a different trick to keep Life from dying out. I put Life in Plane0, and I put Vote in Plane2, with an echo of Vote in Plane3. By

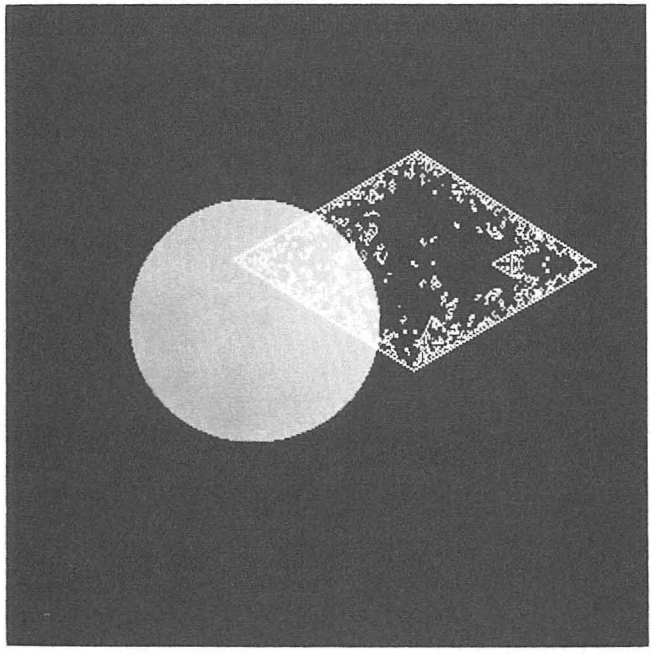

Plate 1.

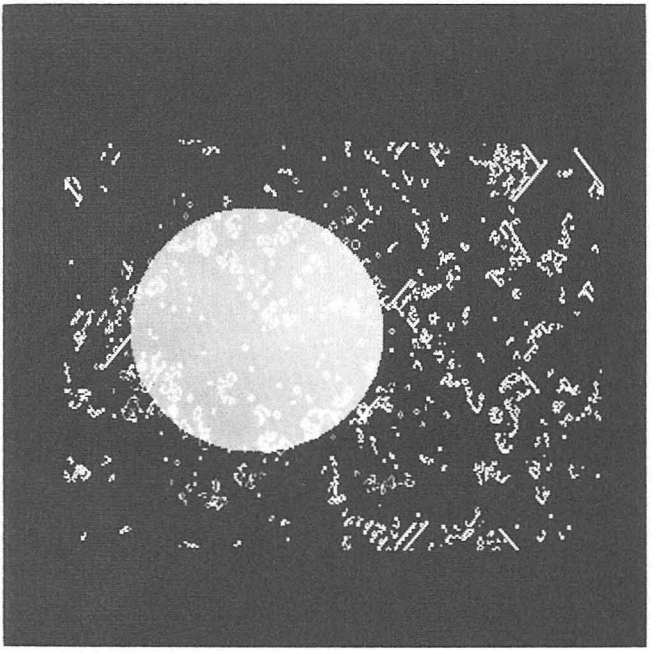

Plate 2.

looking where planes 2 and 3 differ, we can detect the shifting edge of the Vote domain. We let the edge cells turn on Life cells. This has the effect of surrounding the Vote domains with activity, somewhat in analogy to the activity found along beaches. Suppose that we define Edge to be Center2 XOR Center3, the Boolean value of NOT (Center2 = Center3). Then our rule has the form:

Life OR Edge → Plane0
Vote → Plane2
Center2 → Plane3

Votelife is not really symbiotic, as Life has no backreaction on the Vote rule. A backreaction can be arranged if we introduce a new voting rule WVote.

9Sum { 0 0 0 1 1 0 0 1 1 1 }

Run by itself, WVote goes immediately to chaos, but if it is run in some spots on the edge of the normal Vote rule, it has the effect of making the edges more jagged and more stable. Now let Change stand for Center0 XOR Center1, and define a symbiotic Votelife rule JVotelife as:

Life OR Edge → Plane0
Center0 → Plane1
IF Change THEN (WVote → Plane2)
ELSE (Vote → Plane2)
Center2 → Plane3

The edges which arise have a strong fractal appearance.

The third rule I want to describe in this section is called Ranch, and it is a kind of synthesis of Brainlife and Votelife. We run and echo Vote in planes 2 and 3 as before, filling the screen with shifting domains of 0s and 1s. We might think of the black regions of Plane2 as ocean and the light regions as land. We use planes 0 and 1 to run either Life (in Plane0) or Brain (in Plane0 with echo in Plane1). A given cell obeys the Brain rule if it is in "ocean" and the Life rule if it is on "land." A little care needs to be taken at the edge, so that the Brain cells can sometimes get ashore and turn on Life cells. In particular, if Edge is again short for (Center2 XOR Center3), we define Edgelife as:

IF Edge THEN (8Sum { 0 1 0 0 1 0 0 0 0 } → Plane0)
ELSE (Life → Plane0)

Now we can go ahead and define Ranch

IF Center2 THEN Edgelife → Plane0
ELSE Brain → Plane0
Center0 → Plane1
Vote → Plane2
Center2 → Plane3

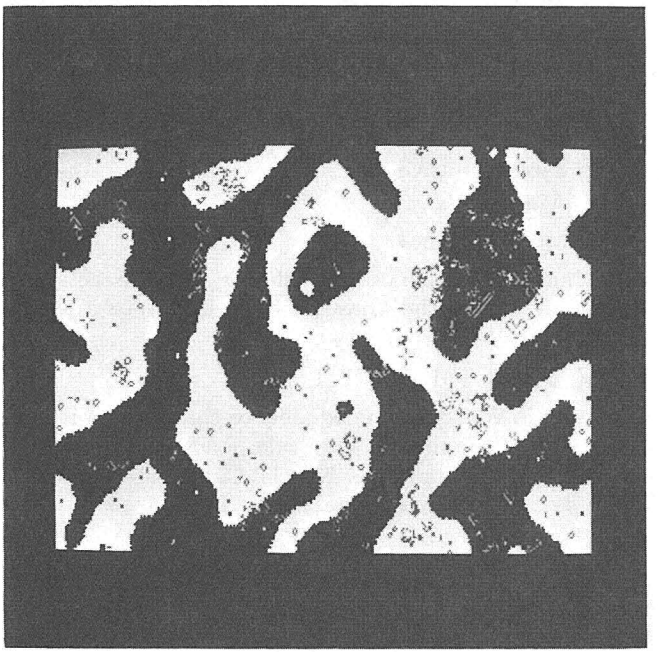

Plate 3.

Ranch is a very interesting rule to watch. If one starts with only a few live Brain cells in the ocean, they will sporadically colonize the banks of the land with Life, sometimes starting backreactions which return new Brain cells to the sea.

Plate 3 shows the rule Ranch. This is a typical pattern evolved some two thousand steps from a random start of half zeroes and half ones. I have run a similar pattern for as many as half a million steps. I found this pattern after searching through a few hundred different random starts. I think of it as an artificial life tide-pool.

In this pattern, the Vote rule establishes two vertical stripes of black and yellow, and these stripes do not go away. (Keep in mind that the screen is treated as a torus, so the two yellow stripes are really one.) The persistence of life Brain cells traveling along the black region of Plate 9 is promoted by the presence of the small manshaped Vote oscillator shown in the center. Under Vote, this oscillator spins about its horizontal axis like an acrobat. I call him "Robotman." Every time a Brain glider hits the Robotman, he gets infected with Life cells which burst back out of him to produce showers of new live Brain cells.

As with Votelife, Ranch has no backreaction on the voting rule. The only

acceptable (noncatastrophic) backreaction I have found is to replace Vote by WVote whenever Center0 OR Center1, but this makes no great change.

I have found one nice rule which sets up a very strong nonlinear feedback between a Life-like rule and a Vote-like rule. This rule, which I call Temple, displays true emergent behavior in that it does some very surprising things. Perhaps the most surprising is the way in which it builds spires on the sides of an ordinary start square, as shown in two slides which Chris Langston lost. If one starts Temple with a random screen, it divides the screen into oppositely striped domains, with some fairly shortlived boiling and sparking along the domain boundaries.

5. Why?

Just about everyone enjoys watching these displays. The simplest explanation is what Jim Crutchfield half-jokingly calls "technomothia": our society conditions people to pay attention to fast-moving bright colors. But after the initial enjoyment, the question often is: "Why are you doing this? What are these good for?"

One aspect of all fast Type 3 CA displays is that here computation is in some sense made visible. The complexity, or program size, of these rules is very low, yet the amount of visual information being produced is large and, in interesting patterns, not readily predictable. In Charles Bennett's terms, these patterns are logically deep [?]. Up to a point, the longer the computer runs, the more information you are getting. Real computational work is being done.

The low-complexity/high-depth character of artificial life CA displays is in some respects the opposite of expert systems AI programs, which typically have extremely long programs (high complexity) and, when actually used, very short run times (low depth). A nice thing about CAs is that one uses a cheap machine to do all the boring work, instead of assigning man-centuries of programming to hapless graduate students.

But this is all at the level of metaphor. How might one practically exploit artificially alive CA patterns for some useful purpose? Phrasing the question in such a way limits the range of interesting answers. A dog, or even a tree, is to a certain extent intelligent. Yet we do not dream of getting a dog or a tree to act as a medical advisor or a missile pilot. We can only exploit the intelligence of a dog or a tree by enjoying the things that they characteristically do (find game, grow a shady umbrella).

It seems certain to me that bigger machines and better breeding will produce CA life of really startling richness. (Margolus and Toffoli are currently building CAM-7, a $1000 \times 1000 \times 1000$ CA board with 16-bit cells.) Yet in their natural course of development, these lifeforms are not particularly "interested" in doing things like speaking English. A rule like Ranch would much rather land gliders on shore to lay eggs.

In a recent paper [?], Danny Hillis, suggests that human speech is a kind of software virus which initially started as songs apes would sing. In his view,

the natural wetware evolution of the apes provided a substrate upon which the rich software of abstract thought memes could fruitfully grow. I think the present function of artificial life breeders is simply to produce systems with rich repertoires of behavior. So many CA rules are computation universal that, when the time comes, many sorts of humanly useful patterns can be taught or programmed in. Given that Brain is almost certainly computation universal, longlived Ranch pattern could presumably be programmed.

Although it has been some years since Conway proved Life to be computation universal, workers such as William Gosper of Symbolics continue refining what Gosper has termed in conversation "the technology of Life." The challenge is to make small and efficient CA patterns which perform useful computations. One can imagine a situation in which computer hardwares were simply very large cellular automaton machines, and users would download the "computer" pattern which they currently needed.

Yet this last idea is probably not the correct one. The essential thing about a CAM is that it is a parallel processor, and it is probably a misallocation of resources to use such a machine to simulate serial computation. As Hyman Hartman pointed out in his talk at ALIFE 87, the best way to "program" parallel computers may simply be to evolve interesting patterns on them, and only then to try to find tasks for which the patterns are appropriate. CAs provide a paradigm for this activity in the same way that the Turing machine is a paradigm for serial computation.

6. Two program listings

As my programs have rather short codelength, it seems valuable to list two of them here so that other CAM-6 workers can run them. The programs are written in an extension of Forth [1,4].

In reading the programs one must understand that the CAM-6 is organized as two halfmachines, CAM-A, which runs planes 0 and 1, and CAM-B, which runs planes 2 and 3. Each halfmachine thinks of its own planes' center cells as CENTER and CENTER′, and the other planes' center cells as &CENTER and &CENTER′. In the rule Temple, the neighborhood assignment is such that one is allowed to look into the neighboring plane to see NORTH′, EAST′, SOUTH′, and WEST′ as well as CENTER′.

People who cannot get access to a CAM-6 should be aware of a new, pure software CA generator called CA Lab. This product is written in C and 8088 assembly language for the PC family by myself and John Walker, co-founder of Autodesk, Inc. CA Lab will be available for $59.95 from Autodesk, Inc., 2320 Marinship Way, Sausalito, CA 94965, starting March 18, 1989. CA Lab can show Ranch and all the cellular automata mentioned in [4], as well as a class of 16-bit rules which solve Laplace's equation in the plane. (I have not tried to make it do Temple yet.) Rules for CA Lab can be programmed via a screen editor or in Basic, Pascal, or C.

6.1 Ranch: A symbiosis of Life and Brain partitioned by Vote

```
NEW-EXPERIMENT
N/MOORE &/CENTERS
                                                          : 8SUM\\
        NORTH EAST SOUTH WEST N.WEST N.EAST S.WEST S.EAST
                                          + + + + + + + + ;
                                                          : 9SUM
                                            8SUM CENTER + ;
                                                          : LIFE
                              8SUM { 0 0 CENTER 1 0 0 0 0 0 } ;
                                                          : BRAIN
                                   8SUM 2 = CENTERS 0 = AND ;
                                                          : VOTE
                                 9SUM { 0 0 0 0 1 0 1 1 1 1 } ;
                                                          : EDGE
                                       &CENTER &CENTER' XOR ;
                                                          :
EDGELIFE
        EDGE IF 8SUM { 0 1 0 0 1 0 0 0 0 } ELSE LIFE THEN ;
CAM-A
                                                          :
RANCH/A
          &CENTER IF EDGELIFE >PLNO ELSE BRAIN >PLNO THEN
                                                CENTER >PLN1 ;
MAKE-TABLE RANCH/A
CAM-B
                                                          :
RANCH/B
                                               VOTE >PLN2
                                             CENTER >PLN3   ;
MAKE-TABLE RANCH/B
```

6.2 Temple: A voting rule driven by Lifey noise

```
NEW-EXPERIMENT
CAM-A N/VONN &/CENTERS
                                                  ; 5SUM
              NORTH EAST SOUTH WEST CENTER + + + + ;
                                                  : 5SUM'
       NORTH' EAST' SOUTH' WEST' CENMTER' + + + + ;
                                                  : TEMPLE/A
   5SUM { 0 0 0 1 1 1 } 5SUM' { 0 0 0 0 0 1 } XOR
                                &CENTER' OR >PLNO
                                    &CENTER >PLN1 ;
MAKE-TABLE TEMPLE/A
CAM-B  N/MOORE &/CENTERS
```

```
                                                    : 8SUM
NORTH EAST SOUTH WEST N.EAST N.WEST S.EAST S.WEST
                                  + + + + + + + + ;
                                                    : 11SUM
            8SUM CENTER &CENTER &CENTER' + + + ;
                                                    : TEMPLE/B
        8SUM { 0 &CENTER CENTER 1 0 0 0 0 0 }
                         CENTER' NOT AND >PLN2
      11SUM { 0 0 0 0  0 0 0 0  0 1 1 1 } >PLN3 ;
MAKE-TABLE TEMPLE/B
```

References

[1] A. Califano, N. Margolus, and T. Toffoli, *CAM-6 User's Guide*, Version 2.1 (Cambridge, MA: MIT Laboratory for Computer Science, 1987).

[2] W.D. Hillis, "Intelligence as an Emergent Behavior," *Daedalus*, **117** (1988) 175–189.

[3] C.G. Langton, "Studying Artificial Life with Cellular Automata," *Evolution, Games, and Learning* (Amsterdam: North Holland, 1986) 120–149.

[4] N. Margolus and T. Toffoli, *Cellular Automata Machines* (Cambridge, MA: MIT Press, 1987).

[5] R. Rucker, *Mind Tools* (Boston, MA: Houghton Mifflin, 1987).

[6] B. Silverman, *The Phantom Fishtank* (Cambridge, MA: Lego-Logo Lab, 1988).

[7] S. Wolfram, *Theory and Applications of Cellular Automata* (Singapore: World Scientific, 1986).