Evolution of Intricate Long-Distance Communication Signals in Cellular Automata Using Genetic Programming

David Andre
Visiting Scholar
Computer Science Dept.
Stanford University
860 Live Oak Ave, #4
Menlo Park, CA 94025 USA
andre@flamingo.stanford.edu

Forrest H Bennett III
Visiting Scholar,
Computer Science Dept.,
Stanford University
Stanford, California 94305
fhb3@slip.net

John R. Koza
Computer Science Dept.,
Stanford University
Stanford, California 94305-9020
koza@cs.stanford.edu
http://www-cs-faculty.stanford.edu/~koza/

Abstract

A cellular automata rule for the majority classification task was evolved using genetic programming with automatically defined functions. The genetically evolved rule has an accuracy of 82.326%. This level of accuracy exceeds that of the Gacs-Kurdyumov-Levin (GKL) rule, all other known human-written rules, and all other rules produced by known previous automated approaches.

Our genetically evolved rule is qualitatively different from other rules in that it utilizes a fine-grained internal representation of density information; it employs a large number of different domains and particles; and it uses an intricate set of signals for communicating information over large distances in time and space.

1. Introduction

Local rules govern the important interactions of many animate and inanimate entities. The study of artificial life often focuses on how the simultaneous execution of a single relatively simple rule at many local sites leads to the emergence of interesting global behavior (Langton 1989). These studies often also deal with the question of how entities that receive information only about their immediate local environment can engage in the long-distance communication necessary to coordinate intricate global behavior. Sometimes, these issues are posed in terms of how complex computations can be performed over great distances in time and space using rules that operate only with data that is nearby in time and space.

2. Automatic Programming of Cellular Automata

Cellular automata are an abstract way of studying and analyzing the simultaneous execution of local rules. Complex overall behavior is often produced by cellular automata as the result of the repetitive application (at each cell in the cellular space) of the seemingly simple transition rules contained in each cell (Burks 1970; Farmer, Toffoli, and Wolfram 1983; Wolfram 1986; Gutowitz 1991).

A cellular space is a uniform array of cells arranged in a certain topological arrangement in a certain number of dimensions. In a *cellular automaton* (CA), each cell in a cellular space is occupied by an identical automaton. The next state of each individual automaton in the cellular space depends on its own current state and on the current states of the other automata in a specified local neighborhood around the individual automaton. The state of each automaton at time 0 is called its *initial condition*.

For a one-dimensional cellular automaton, the cellular space is a linear arrangement of identical automata. The next state of each individual automaton might depend, for example, on the current state of that automaton and the current states of its six neighbors at distances up to 3. We denote these seven neighbors as `X` (for the automaton at the center), `W` (the adjacent automaton to the west), `E` (the adjacent automaton to the east), `WW` (the automaton at distance 2 to the west), `EE`, `WWW`, and `EEE`. A cellular space is said to have *periodic boundary conditions* when the cellular space is toroidal. If the automaton located in each cell has only two states, the state-transition function of the automaton is a Boolean function of its own current state and the states of its neighbors at a specified distance.

It is extremely difficult, in general, to design a single state-transition rule that, when it operates in each cell of the cellular space, produces a desired behavior.

Genetic algorithms (Holland 1975) operating on fixed-length character strings have been successfully used to evolve the initial conditions and state-transition rules for cellular automata (Meyer, Richards, and Packard 1991). Genetic programming (Koza 1992, 1994a, 1994b; Koza and Rice 1992) has been used for automatic programming of cellular automata randomizers (Koza 1992).

3. The Majority Classification Problem

The majority classification problem is one vehicle for exploring how complex calculations can be performed over large areas using rules of interaction that operate over a relatively small distance. In one commonly studied version of this problem, there is a one-dimensional linear arrangement of 149 two-state automata whose update rule operates on information within a distance of three. The initial states of the 149 automata (called the *initial configuration*) are the inputs to the calculation. If all 149 automata relax to a common state (0 or 1) after a certain amount of time, the common state is considered to be the binary output of the calculation. For the majority

classification task, 149 0's constitute the correct answer if a majority of the 149 initial bits are 0 and a 149 1's are correct if a majority of the bits are 1. Thus, to solve this problem, a seven-argument Boolean transition rule must be found such that when it is situated at all 149 cells of a one-dimensional cellular automaton, the automaton converges to the correct configuration of 149 0's or 149 1's after 600 time steps. The fitness of a rule is measured by its ability to correctly do this computation for a random initial configuration.

It is difficult to construct a two-state, seven-neighborhood cellular automata rule that performs this majority classification task reasonably well on a 149-bit input configuration. Seven bits cannot store an integer as large as 149 (if, for example, one were storing and transmitting a signal communicating a locally observed excess of 0's or 1's). It is unknown whether a perfect solution to this problem exists.

The difficulty that human programmers have had with this task is indicated by its history. In 1978, Gacs, Kurdyumov, and Levin developed a two-state, seven-neighbor rule for the purpose of studying reliable computation under random perturbations. This Gacs-Kurdyumov-Levin (GKL) rule performs the majority classification task reasonably well. The GKL rule is successful on 8l.6% of the inputs consisting of a configuration of 149 bits (each chosen independently with 50% probability). Gonzaga de Sa and Maes (1992) showed that this system does indeed relax to a common state.

The majority classification task has been the subject of extensive study (Mitchell, Hraber, and Crutchfield 1993; Das, Mitchell, and Crutchfield 1994; Mitchell, Crutchfield, and Hraber 1994; Crutchfield and Mitchell 1995; Das, Crutchfield, Mitchell, and Hanson 1995; Mitchell 1996).

Lawrence Davis (1995) cleverly modified the GKL rule and created a rule that achieved slightly better accuracy than the GKL rule. His rule has an accuracy of about 81.8%. In 1993, Rajarshi Das (1995) created another rule that achieved an accuracy of about 82.178%. Since several of these rules have not been previously published, we show them in table 1. The last row in this table shows the genetically evolved rule described later in this paper. The 128 bits are presented in the natural order that they would appear in a state transition table starting with state 0000000 and ending at state 1111111.

Das, Mitchell, and Crutchfield (1994) evolved rules using a version of the genetic algorithm operating on fixed-length strings. These evolved rules sometimes exhibited qualitatively the same behavior as the GKL rule; however, none of the rules evolved using the genetic algorithm operating on fixed-length strings were as accurate as the original GKL rule. As Das, Mitchell, and Crutchfield (1994) reported, the genetic algorithm usually found only relatively uninteresting block-expanding rules that score in the range of 65-70% accuracy. The best result from their studies had 76.9% accuracy. This reported performance may have been the consequence of factors such as the small population size employed.

Land and Belew (1995) suggested that the standard representation of the cellular automata rule as a chromosome string of length 128 used in most previous work in evolving one-dimensional cellular automata using the genetic algorithm may hinder evolution. They suggest that higher-level representations (such as condition-action pairs) may aid the evolutionary process because of the higher degree of locality in the condition-action pairs.

Given Land and Belew's (1995) findings, the tree representation employed by genetic programming seems well suited for this task because it permits the size and shape of the ultimate solution to undergo evolution. The search strategy employed by genetic programming (and the genetic algorithm) is also important for this problem. Although many Boolean problems can be trivially solved by hillclimbing methods in a variety of representations, such methods work only when the fitness cases (i.e., the 2^7 lines in the truth table) are independent. When a Boolean function is used for a cellular automata rule, there is no correct answer for a given set of inputs that is independent from the answers for the other inputs. Also, genetic programming supports automatically defined functions, whereas the conventional genetic algorithm operating on fixed-length strings does not have a similar facility for exploiting regularities, symmetries, homogeneities, and modularities of the problem environment.

4. Preparatory Steps

The runs reported here used standard genetic programming with automatically defined functions (Koza 1994a) as summarized in table 2. The problem (coded in ANSI C) was run on a medium-grained parallel Parystec computer system consisting of 64 Power PC 601 80 MHz processors arranged in a toroidal mesh with a host PC Pentium type computer.

Table 1 The succession of "best" cellular automata rules for the majority classification task.

Rule	State Transitions
GKL 1978	00000000 01011111 00000000 01011111 00000000 01011111 00000000 01011111 00000000 01011111 11111111 01011111 00000000 01011111 11111111 01011111
Davis 1995	00000000 00101111 00000011 01011111 00000000 00011111 11001111 00011111 00000000 00101111 11111100 01011111 00000000 00011111 11111111 00011111
Das 1995	00000111 00000000 00000111 11111111 00001111 00000000 00001111 11111111 00001111 00000000 00000111 11111111 00001111 00110001 00001111 11111111

GP 1995	00000101 00000000 01010101 00000101 00000101 00000000 01010101 00000101 01010101 11111111 01010101 11111111 01010101 11111111 01010101 11111111

Table 2 Tableau for the majority classification problem for one-dimensional cellular automata.

Objective:	Find a seven argument Boolean function that performs the majority classification problem for a 149-width one-dimensional cellular automata.
Architecture of the overall program with ADFs:	One result-producing branch and one 2-argument automatically defined function, `ADF0`, and one 3-argument automatically defined function, `ADF1`. `ADF1` can refer to `ADF0`.
Terminal set for the RPB:	`X`, `E`, `EE`, `EEE`, `W`, `WW`, and `WWW`.
Function set for the RPB:	`AND`, `OR`, `NAND`, `NOR`, `NOT`, `IF`, `XOR`, `ADF0`, and `ADF1`.
Terminal set for `ADF0`:	`X`, `E`, `EE`, `EEE`, `W`, `WW`, `WWW`, `ARG0`, and `ARG1`.
Function set for `ADF0`:	`AND`, `OR`, `NAND`, `NOR`, `NOT`, `IF`, and `XOR`.
Terminal set for `ADF1`:	`X`, `E`, `EE`, `EEE`, `W`, `WW`, `WWW`, `ARG0`, `ARG1`, and `ARG2`.
Function set for `ADF1`:	`AND`, `OR`, `NAND`, `NOR`, `NOT`, `IF`, `XOR`, and `ADF0`.
Fitness cases:	• 1,000 149-bit initial configurations were used as in-sample fitness cases. These initial configurations were created randomly, with no bias (i.e., 0 and 1 each have an independent 50% probability of being chosen). Thus, the distribution of densities of the initial state vectors was a binomial distribution centered at 0.50. • The out-of-sample fitness cases consisted of 1,000,000 (and later 10,000,000 and 15,000,000) similarly created initial configurations.
Raw fitness:	1,000 minus the number of fitness cases for which the system relaxes to the correct configuration after 600 time steps.
Standardized fitness:	1,000 minus raw fitness.
Hits:	Raw fitness.
Parameters:	• Population size, M , is 51,200 (64 times 800). • Maximum number of generations to be run, G, is 51. • 89% crossovers, 10% reproductions, and 1% mutations were used on each generation. • A maximum of 500 points (functions and terminals) for the result-producing branch and 250 points for each automatically defined function was allowed. • Structure-preserving crossover with branch typing was used. • The other parameters for controlling the runs of genetic programming were the default values specified in Koza (1994a).

The so-called *distributed genetic algorithm* or *island model* for parallelization was used. That is, subpopulations (*demes*) were situated at the processing nodes of the system. Population size was Q = 800 at each of the D = 64 demes for a total population size of 51,200. The initial random subpopulations were created locally at each processing node. Generations were run asynchronously on each node. After a generation of genetic operations was performed locally on each node, four boatloads, each consisting of B = 3% (the migration rate) of the subpopulation (selected on the basis of fitness) were dispatched to each of the four toroidally adjacent nodes. Details of the parallel implementation of genetic programming can be found in Andre and Koza 1996.

5. Emergent Properties of the Most Successful Run

We made five runs of genetic programming on this problem. Each run produced hundreds of individuals that behaved in a manner reminiscent of the GKL rule. Each of the five runs produced numerous individuals that score well above the best accuracy of 76.9% produced by the genetic algorithm operating on fixed-length character strings.

The best-of-run individual from one of these five runs scores a higher accuracy than any other known rule (table 1), including the GKL rule, all other known human-written rules, and all rules produced by all known previous automated approaches.

An examination of the most successful run illustrates the emergence of many interesting entities prior to the creation of the best individual.

The best individual program from the initial random generation indiscriminately classifies all fitness cases as having a majority of 1's . Since 525 of the 1,000 randomly created fitness cases on this particular run happened to have a majority of 1's , this best of generation 0 scores 525 hits.

The best individual of generation 1 scores 650 hits. The activity of a one-dimensional cellular automata can be presented as a two dimensional grid in which the top horizontal row contains the states (0 or 1) of the 149 automata at a time 0 (i.e., the initial configuration of the system) and in which each successive row represents the states of the 149 automata at successive time steps. Figure 1 shows a small part of such a diagram for this individual. In the space-time diagram, we see large areas dominated by solid blocks of repeated simple regular patterns, called *domains*, where a given domain is specified by a regular expression. The two domains shown in figure 1 are the domain denoted by the regular expression 1^* and the domain denoted by the expression 0^*. The space-time diagrams in this paper display 1's as black, and 0's as

white. Thus we call the 1^* domain black (designated by (**B**)) and the 0^* domain white (designated by (**W**)).

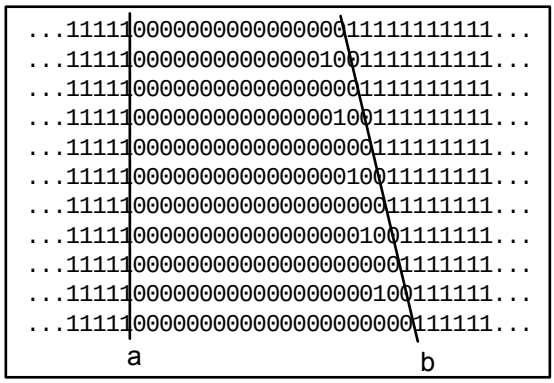

Figure 1. Eleven time steps of the partial space-time behavior of the best individual of generation 1 on one fitness case showing the spread of the zero domain.

The domains of this generation 1 individual interact in various ways. When a domain consisting of a solid block of 1's is to the left of a domain consisting of a solid block of 0's , the interface consists of ...111000... and the corresponding positions at the next time step also consist of ...111000... (where the underlined digit represents the same automaton). Because these particular two domains do not move left or right over time, this particular domain interface is said to have a velocity of zero.

Following Das, Mitchell, and Crutchfield (1994), we describe such an interface between two domains as a particle, denoted P(xy), where x is the domain on the left and y the domain on the right. Particles are one of the ways that information is communicated across large distances in time and space in a cellular space.

In figure 1, line a shows the path of the P(BW) particle with a velocity of 0 and line b shows the path of the P(WB) particle with a velocity of 1/2.

This evolved automata from generation 1 overpredicts a majority of 1's and underpredicts a majority of 0's . This individual is an example of what Mitchell et al. (1993) called a *block-expanding rule* (i.e. a rule that converges to a state unless a sufficiently large block of adjacent or nearly adjacent instances of the opposite state exists in the input).

The best program of generation 6 scores 706 hits on the 1,000 in-sample fitness cases. A portion of its space-time behavior is shown in figure 2. The behavior of this individual is similar to the individual of generation 1. However, it scores slightly better because it has modified the conditions under which it expands a block. The interaction between the domain of zeros on the left and ones on the right is quite complex in this individual, and the interface between the two domains is quite large. The particle $P_{(WB)}$ travels at a velocity of 2/7.

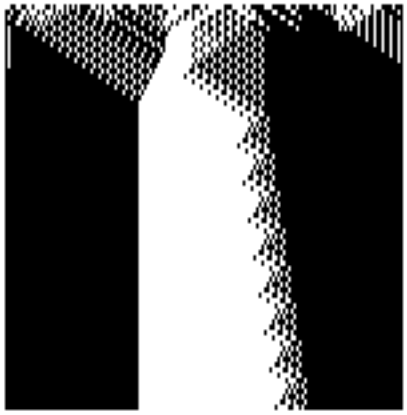

Figure 2. Partial space-time behavior of the best individual of generation 6 on one fitness case. Black represents 1's , white 0's .

The best individual of generation 15 scores 815 hits. Its behavior (see figure 3) is somewhat similar to that of the GKL rule, although it scores less than 80% on out-of-sample fitness cases. Like the GKL rule (figure 7), this rule separates white on the left from black on the right with a gray domain that grows into both black and white. When black is to the left of white, a zero-velocity particle characterizes their interaction. The combination of these two concepts yields the basic mechanism that both GKL and this generation 15 individual use to compute global measures of density. If the domain of all white is larger, it will win out over black, and vice versa.

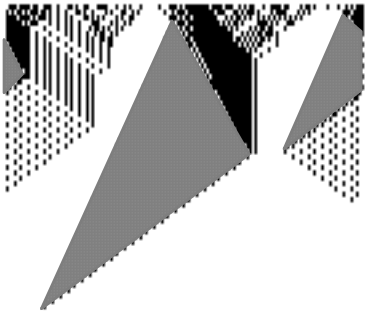

Figure 3. Space-time behavior of the best individual of generation 15 on a typical initial state vector.

Although similar to GKL, the rule of generation 15 has several new gray domains that are not quite the same as the gray domain in the GKL rule's behavior. The new domains interact, as can be seen in figure 4. In this example, the zero and one domains do not enter into the computation until the very end – all of the primary computation is performed by the new gray domains. At this point, however, the new gray domains do not interact particularly well – the rule fails often when it performs computation with the new gray domains.

The best individual of the run emerged on generation 17. The 36-point result-producing branch of the evolved Boolean expression is shown below:

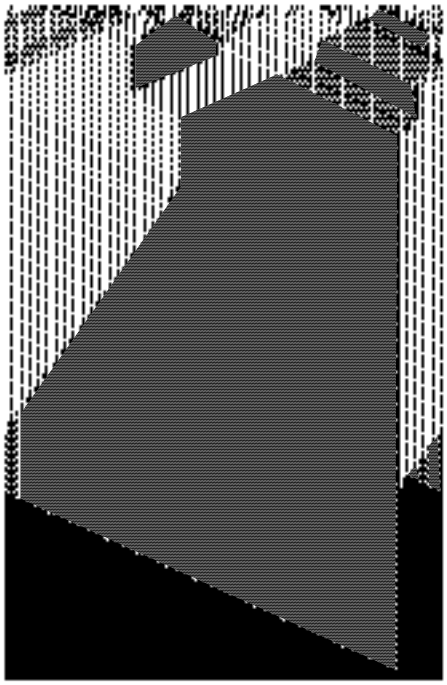

Figure 4. Space-time behavior of the best individual of generation 15 where there are no all black or all white domains until the end.

```
(nand (xor (or (adf0 X (not WWW)) (not WWW))
(adf0 (adf0 E X) (or E W))) (or (nand (if EEE
E X) (if EEE W (and EE WWW))) (adf1 (nand W
E) (xor WWW W) E))).
```

The 10-point automatically defined function ADF0 is

```
(if (nor (or X X) (nor arg1 arg0)) EEE WWW).
```

The 44-point automatically defined function ADF1 is

```
(xor W (xor (adf0 (adf0 (if (or (not W) (adf0
WWW arg2)) (xor (adf0 WWW X) (nor WW WWW))
(nand (xor X X) (or X WW))) (nor (or (if arg2
EEE arg2) (nor arg2 arg1)) (not (xor W E))))
(xor W (not arg2))) EEE)).
```

This rule scores 824 hits on the 1,000 in-sample fitness cases, and scores 82.4% accuracy over 100,000 out-of-sample cases. It scores 82.326% accuracy over 10,000,000 additional cases. This accuracy is slightly better than the score of the GKL rule. We are aware of two human-coded rules that also are better than the GKL rule, the Davis rule (1995) and the Das rule (1995). All four rules are shown in table 1. Table 3 shows the results of testing (on the same out-of-sample fitness cases) the three human-written rules, the best rule produced by the genetic algorithm, and the best rule evolved by genetic programming. As can be seen, the best rule evolved by genetic programming is slightly better than all the other rules. Non-parametric c2 tests with one degree of freedom were performed; the probability(p) that the pairwise differences between the best rule evolved by genetic programming and each of the other three rules were attributable to chance was less than 0.001. (i.e. the differences are statistically significant, $p < 0.001$).

Table 3. Out-of-sample comparison.

Rule	Accuracy	Number of test cases
Best rule evolved by genetic algorithm (Das et al 1994)	76.9%	10^6
GKL human-written	81.6%	10^6
Davis human-written	81.8%	10^6
Das human-written	82.178%	10^7
Best rule evolved by genetic programming	82.326%	10^7

6. The Best Rule Evolved by Genetic Programming

Why does the genetically evolved rule score better than the best known rules written by humans? First, the evolved rule has more domains than the GKL rule. These eleven domains, shown in table 4, classify the density of 1's into finer levels of gray than do the domains found in the GKL rule's behavior as discussed by Das, Mitchell, and Crutchfield (1994). The domains in the GKL rule's behavior are black, white, and checkerboard gray, corresponding to the regular expression $(10)^* \approx (01)^*$ (Das, Mitchell, and Crutchfield 1994). In addition, the evolved rule's computation uses a larger number of particles. The genetically evolved rule discovered 10 particles which are identical to the particles (Das, Mitchell, and Crutchfield 1994) in the GKL rule; however, it also discovered at least 40 additional particles which involve the new gray domains. All the new particles have velocity 0, +3, or -3.

Figure 5. Space-time diagram showing the behavior of the best evolved rule. The differences between the shades of gray are crucial, especially in the area pointed to by the arrow, as otherwise the automata would not converge in this example.

Having more domains would be purely a superficial difference if the new domains did not participate in the computation. However, it appears that the new gray domains are crucial to the computations being performed by the genetically evolved rule. In the behavior shown in figure 5, for example, the distinct behavior of the new grays is critical to the success of the rule. If all the grays acted identically, then the automata would not converge in this example. The interactions of the new gray domains in the area indicated by the arrow allow the white domain to escape and encroach to the left.

In addition, there are time periods in the behavior of the evolved rule where none of the GKL-like domains exist; the computation is carried out entirely by the new gray domains. Figure 6 shows such a space-time diagram.

Figure 6. The behavior of the best evolved individual on one set of initial states.

Even though it is apparent that the new gray domains take part in the computation, one might question whether they account for any part of the difference in score between the best evolved rule and the GKL rule. However, the behavior of both rules shown in figure 7 indicates that the finer levels of gray do give the evolved rule some advantage. On the left of the initial state vector, there are two fairly large regions of white (0's) separated by a region of black (1's) marked in the diagrams by the arrows. Under the GKL rule, the black area is eliminated by the two adjacent white domains. Under the evolved rule, these two white areas get classified as light gray domains, and the black area becomes a dark gray domain (all shifted to the right slightly). Because it can make use of extra domains, the evolved rule can keep the information that there is a significant density of black (1's) in the vicinity of the arrow until more global information can resolve the local dispute.

The basic mechanism of the evolved rule is similar to that of the GKL rule in that there is a race among the black, white, and gray areas (Das, Mitchell, and Crutchfield 1994). Figure 8 shows a space-time diagram of the behavior of the evolved rule.

In the evolved rule, a white domain (**W**) is separated on the right from a black domain (**B**) by a growing domain of checkerboard gray (**6**). The white domain (**W**) is separated on the left from the black domain (**B**) by one or both of the separator domains, (**3**) and (**8**), which are domains whose interactions with either white or black have zero velocity. In the case shown in figure 8, the (**B**) domain is larger, and the gray (**6**) domain cuts off the white domain (**W**), allowing the black domain (**B**) to break through. In this case, the extra gray domains of (**7**) and (**9**) are not important, but they play a role in other fitness cases.

The mechanism discussed above is sufficient for many cases, but fails to handle the case when the black and white domains are of approximately equal size. In this circumstance, the gray (**6**) domain cuts off both the white and black domains. In the behavior of the GKL rule, when there is such a 'tie', after a brief transition, the black and white domains seem to 'swap' places, separated again by a new growing gray domain.

In the behavior of the best evolved rule such collisions occur slightly differently and yield different behavior after the collision. Under the evolved rule, the black (**B**) and white (**W**) domains are separated by a potentially larger distance by one or both of the separator domains (**8**) and (**3**). If there is a tie, new domains emerge after the collision, and there is no immediate gray separator domain.

Table 4 The domains of the best evolved rule.

Regular Domain	**Domain Name**	**Color**
0*	**W**	White
(000001)*≈(100000)*≈(010000)*≈(001000)*≈(000100)*≈(000010)*	**1**	Very Light Gray
(000101)*≈(100010)*≈(010001)*≈(101000)*≈(010100)*≈(001010)*	**2**	Light Gray
(001)*≈(100)*≈(010)*	**3**	Light Gray
(001101)*≈(100110)*≈(010011)*≈(101001)*≈(110100)*≈(011010)*	**4**	Gray
(x001011)*≈(100101)*≈(110010)*≈(011001)*≈(101100)*≈(010110)*	**5**	Gray
(01)*≈(10)*	**6**	Gray
(011101)*≈(101110)*≈(010111)*≈(101011)*≈(110101)*≈(111010)*	**7**	Dark Gray
(011)*≈(101)*≈(110)*	**8**	Dark Gray
(011111)*≈(101111)*≈(110111)*≈(111011)*≈(111101)*≈(111110)*	**9**	Very Dark Gray
1*	**B**	Black

The new gray domains of (**7**) and (**9**) appear to represent the continuation of the black (**B**) domain, whereas the (**2**) and (**1**) domains appear to represent the continuation of the white (**W**) domain. Thus, when there is a tie in the evolved rule, the original domains of black and white disappear completely, and the computation is carried out entirely by the gray domains.

In the circumstance shown in figure 9, the black (**B**) and white domains (**W**) are separated by the (**3**) domain. When the checkerboard gray (**6**) domain cuts off the (**B**) domain, a new domain (**1**) is created that encroaches on both the (**6**) domain and the (**3**) separator domain. As it happens, the (**6**) domain cuts off the white (**W**) domain just as the particle $P_{(13)}$ reaches the (**W**) domain. As a result, the dark gray domain (**7**) begins to encroach on the gray (**6**) domain and creates a vertical wall with the (**1**) domain. Then, when the dark gray domain (**7**) reaches the very light gray domain (**1**) on the right, the gray (**5**) domain is formed.

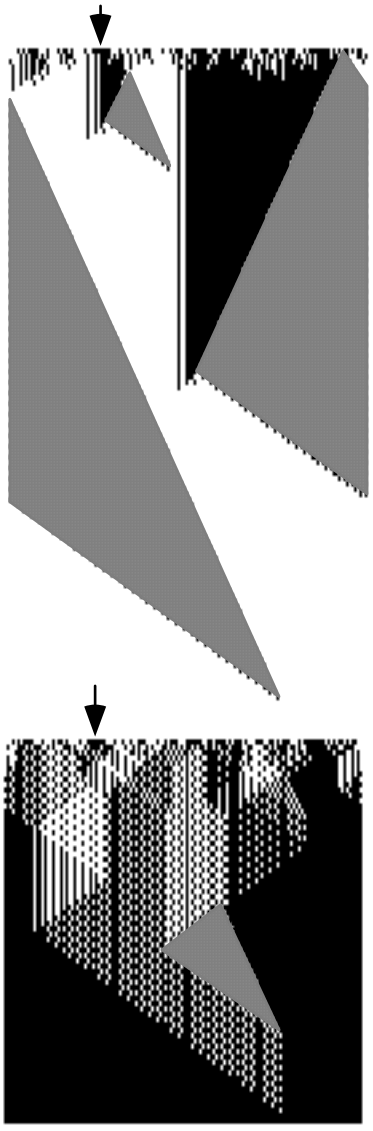

Figure 7. The space-time behavior of the evolved rule (bottom) and the GKL rule (top) on the same initial state vector. The ability of the evolved rule to classify domains of the input as intermediate gray values allows it to avoid the 'round-off' error that the GKL makes.

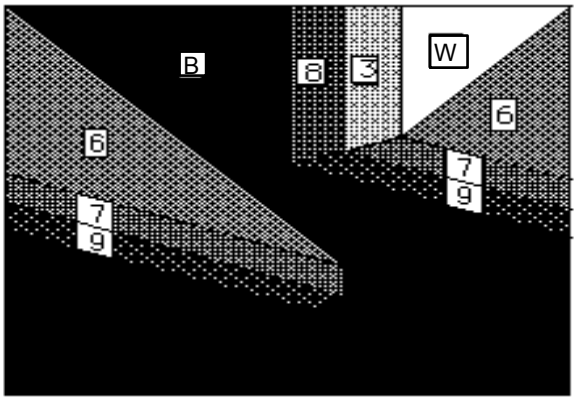

Figure 8. Abstracted Space-Time domain diagram for the best evolved rule.

This domain (**5**) serves a similar purpose as the checkerboard gray domain (**6**), in that it separates a dark domain (**7**) from a very light domain (**1**). When the gray domain (**5**), which grows to the left into the (**7**) domain, reaches the very light gray domain (**1**), the very light gray domain (**1**) begins to expand to the right. As might be expected, when the very light gray domain (**1**) hits a very light gray domain (**1**), a pure white domain (**W**) is created.

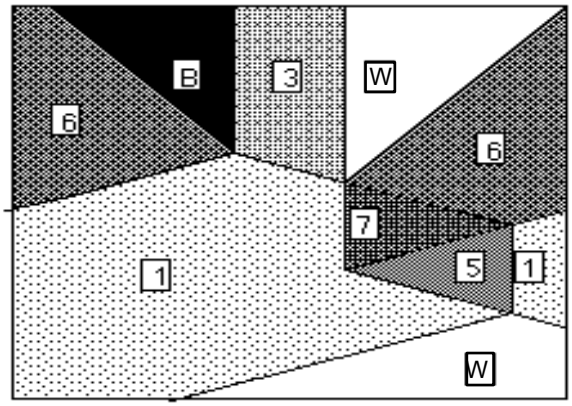

Figure 9 The best evolved rule's abstracted space-time domain behavior on a tie where the answer is white (W).

One problem that a rule utilizing zero-velocity particles can have is that the initial state vector could either be in a state or relax to a state where convergence does not take place. If the automata either starts in or reaches a state where the automata is filled with domains that have zero-velocity interactions with the adjacent patterns, the automata cannot converge. Thus, one potential downside to the evolved rule is that because it utilizes more domains in its computation that have zero-velocity particles, it may be more likely to be trapped in a non-convergent steady-state. We found only one such input pattern in our testing (see figure 10). Given that the evolved rule scores better than any other known rule, however, it is possible that the occasional non-convergence represents a strategic tradeoff rather than a defect.

7. Conclusions

The success of genetic programming on this problem suggests that genetic programming might be useful for other problems where emergent computation is sought.

A cellular automata rule for the majority classification task was evolved using genetic programming with automatically defined functions. The genetically evolved rule has an accuracy of 82.326%. This accuracy exceeds that of the Gacs-Kurdyumov-Levin (GKL) rule, all other known human-written rules, and all other known rules produced by previous automated approaches. The genetically evolved rule is qualitatively different from these rules. The genetically evolved rule utilizes a very fine internal representation of density information; it employs a large number of different domains and particles; and it uses an intricate set of signals for communicating information over large distances in time and space.

Figure 10. The behavior of the evolved rule on an initial state vector where the automata does not converge.

Acknowledgements

The authors gratefully thank Lawrence Davis, Melanie Mitchell, and James Crutchfield for helpful discussions and for reviewing a draft of this paper and Simon Handley for reviewing a draft of this paper.

Bibliography

Andre, David and Koza, John R. 1996. Parallel genetic programming: A scalable implementation using the transputer architecture. In Angeline, Peter J. and Kinnear, Kenneth E. Jr. (editors). *Advances in Genetic Programming 2*. Cambridge, MA: MIT Press.

Burks, Arthur W. 1970. *Essays on Cellular Automata*. Urbana, IL: University of Illinois Press.

Crutchfield, J. P. and Mitchell, Melanie. 1995. The evolution of emergent computation. *Proceedings of the National Academy of Sciences, USA*. 92 (23).

Das, Rajarshi, Mitchell, Melanie, and Crutchfield, J. P. 1994. A genetic algorithm discovers particle-based computation in cellular automata. In Davidor, Yuval, Schwefel, Hans-Paul, and Maenner, Reinhard (editors). 1994. *Parallel Problem Solving from Nature - PPSN III.* (Lecture Notes in Computer Science, Volume 866). Berlin: Springer-Verlag. 344-353.

Das, Rajarshi, Crutchfield, J. P., Mitchell, Melanie, and Hanson, J. E. 1995. Evolving globally synchronized cellular automata. In Eshelman, Larry J. (editor). *Proceedings of the Sixth International Conference on Genetic Algorithms*. San Francisco, CA: Morgan Kaufmann Publishers.

Das, Rajarshi. 1995. Personal communication.

Davis, Lawrence. 1995. Personal communication.

Farmer, Doyne, Toffoli, Tommaso, and Wolfram, Stephen (editors). 1983. *Cellular Automata: Proceedings of an Interdisciplinary Workshop, Los Alamos, New Mexico, March 7-11, 1983*. Amsterdam: North-Holland Physics Publishing. Also in *Physica D*, volume 10.

Gacs, P., Kurdyumov, G. L., and Levin, L. A. 1978. One dimensional uniform arrays that wash out finite islands. *Problemy Peredachi Informatsii*. 12(1978) 92 – 98.

Gonzaga de Sa, P. and Maes. C. 1992. The Gacs-Kurdyumov-Levin automaton revisited. *Journal of Statistical Physics*. 67(3/4) 507–522.

Gutowitz, Howard (editor). 1991. *Cellular Automata: Theory and Experiment*. Cambridge, MA: MIT Press. Also in *Physica D* 1990.

Holland, John H. 1975. *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press. The 1992 second edition was published by MIT Press.

Koza, John R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* Cambridge, MA: MIT Press.

Koza, John R. 1994a. *Genetic Programming II: Automatic Discovery of Reusable Programs.* Cambridge, MA: MIT Press.

Koza, John R. 1994b. *Genetic Programming II Videotape: The Next Generation*. Cambridge, MA: MIT Press.

Koza, John R., and Rice, James P. 1992. *Genetic Programming: The Movie*. Cambridge, MA: MIT Press.

Land, Mark and Belew, Richard K. 1995. In McDonnell, John R., Reynolds, Robert G., and Fogel, David B. (editors). *Evolutionary Programming IV*: *Proceedings of the Fourth Annual Conference on Evolutionary Programming*. Cambridge, MA: MIT Press. Pages 403–434.

Langton, Christopher G. (editor). 1989. *Artificial Life, Santa Fe Institute Studies in the Sciences of Complexity*. Volume VI. Redwood City, CA: Addison-Wesley.

Meyer, Thomas P., Richards, Fred C. and Packard, Norman H. 1991. Extracting cellular automaton rules directly from experimental data. In Gutowitz, Howard (editor). *Cellular Automata: Theory and Experiment*. Cambridge, MA: MIT Press.

Mitchell, Melanie. 1996. *An Introduction to Genetic Algorithms*. Cambridge, MA: MIT Press.

Mitchell, Melanie, Crutchfield, J. P., and Hraber, P. T. 1994. Dynamics, computation, and the "edge of chaos": A re-examination. In G. Cowan, D. Pines, and D. Melzner (editors). *Complexity: Metaphors, Models, and Reality*. Santa Fe Institute Studies in the Sciences of Complexity, Proceedings, Volume 19. Reading, MA: Addison-Wesley.

Mitchell, Melanie, Hraber, P. T., and Crutchfield, J. P. 1993. Revisiting the edge of chaos: Evolving cellular automata to perform computations. *Complex Systems*. (7) 89-130.

Wolfram (editor). 1986. *Theory and Applications of Cellular Automata*. Singapore: World Scientific.