Evolving Cellular Automata to Perform Computations: Mechanisms and Impediments

Melanie Mitchell[1], James P. Crutchfield[2], and Peter T. Hraber[1]

Santa Fe Institute Working Paper 93-11-071

Submitted to *Physica D*
October 18, 1993

Abstract

We present results from experiments in which a genetic algorithm was used to evolve cellular automata (CAs) to perform a particular computational task—one-dimensional density classification. We look in detail at the evolutionary mechanisms producing the GA's behavior on this task and the impediments faced by the GA. In particular, we identify four "epochs of innovation" in which new CA strategies for solving the problem are discovered by the GA, describe how these strategies are implemented in CA rule tables, and identify the GA mechanisms underlying their discovery. The epochs are characterized by a breaking of the task's symmetries on the part of the GA. The symmetry breaking results in a short-term fitness gain but ultimately prevents the discovery of the most highly fit strategies. We discuss the extent to which symmetry breaking and other impediments are general phenomena in any GA search.

[1]Santa Fe Institute, 1660 Old Pecos Trail, Suite A, Santa Fe, New Mexico, U.S.A. 87501.
Email: mm@santafe.edu, pth@santafe.edu

[2]Physics Department, University of California, Berkeley, CA, U.S.A. 94720.
Email: chaos@gojira.berkeley.edu

1. Introduction

Cellular automata (CAs) are spatially-extended discrete dynamical systems whose architecture has many desirable features for a large class of parallel computations. In scientific modeling applications, CAs have been used to simulate, for example, magnetic spin systems [15, 89], fluid dynamics [21, 27], chemical oscillations [52, 63], crystal growth [51, 64], galaxy formation [30], stellar accretion disks [76], dynamics in cytoskeletal lattices [81], and the formation of biological patterns (e.g., the intricate fractal patterns seen on mollusk shells [11], or vertebrate pigment patterns [91]). Common to all these modeling applications is the belief that CAs can capture essential features of physical systems in which large-scale behavior arises from the collective effect of large numbers of locally interacting simple components. In engineering applications, CAs have been used to perform, among other things, parallel formal-language recognition [69, 80] and a range of image-processing tasks [54, 71, 72, 75, 82, 92]. There are many other potential engineering applications of CAs, such as forecasting, spatio-temporal noise-reduction, the discovery of coherent structures in data, texture detection, and so on.

The massive parallelism and local connection architecture of CAs, as well as their capacity for resistance to error and noise, means that hardware implementations have the potential for extremely fast and reliable computation that is robust to noisy input data and component failure [28]. The standard approach to parallel computation is to split up a problem into independent modules that are then parceled out to different processors, solved simultaneously, and the piecewise solutions recombined. In contrast, a CA performs computation in a distributed fashion on a spatially-extended lattice. CAs suggest new ways of parallelizing problems that are hard to split up and parcel out. Recent work on CAs has yielded much new insight into the mechanisms by which complex behavior can arise in nonlinear spatially-extended systems with local interactions (e.g., see [23, 34, 87, 90]). However, little is known about how to harness this complex behavior to perform useful computation, since in general it is hard to predict, much less design, the behavior of such systems. The study of CA-based information processing is a case of the general problem of harnessing the computational power of spatially-extended dynamical systems. The difficulty of designing CAs to have desired behavior or to perform a particular task has up to now severely limited their applications in science and engineering, and for general computation. Finding a way to automate the design of CAs would thus have great significance for a number of fields.

In this paper we describe research on using genetic algorithms (GAs) to evolve CAs to perform computations. GAs are search and optimization methods based on ideas from natural genetics and evolution [19, 31, 41, 55]. A GA works on populations of "chromosomes" that represent candidate solutions to a given problem, applying "genetic" operators such as fitness-based reproduction, crossover, and mutation to members of the population over a number of "generations". GAs have become increasingly popular in recent years in machine learning and other disciplines because of their utility in a range of applications. Examples of application areas include engineering design (e.g., aircraft design [10], circuit design [79], and engine-turbine design [70]), operations research (e.g., [4, 35]), automatic programming (e.g., [39, 45]), neural-network design (e.g., [8, 12, 37, 57, 61, 33]), robot control (e.g., [22, 38, 18]), and molecular biology (e.g., DNA sequence assembly [68] and protein-structure prediction [17, 78, 88]). GAs have also been used as scientific models of evolutionary processes in natural systems. Examples include models of economic systems (e.g., [3, 43]), models of the immune system (e.g., [26]), models of ecological phenomena such as biological arms races, host-parasite co-evolution, symbiosis, and resource flow in ecologies (e.g., [5, 6, 13, 14, 39, 41, 42, 44, 49, 50, 66, 73, 74, 84]), models of phenomena in population genetics such as the evolution of recombination (e.g., [9, 24, 53, 77]), and models of the interaction between evolution and learning (e.g., [1, 2, 7, 25, 40, 62, 56, 67, 85, 86]).

The goals of our research are: (1) to better understand the ways in which CAs can perform computations; (2) to learn how best to use GAs to evolve computationally useful CAs; and (3) to understand the mechanisms by which evolution—as modeled by a GA—can create complex, coordinated global behavior in a system consisting of many locally interacting simple parts. CAs are perhaps the simplest examples of such systems. In nature, evolution has resulted in high levels of computational capability within much more complicated systems—a preeminent example being the human nervous system.

In this paper we analyze the GA's behavior in evolving one-dimensional CAs to perform a particular computational task. We investigate both the mechanisms underlying the GA's performance and the impediments it faces in finding CAs that achieve high performance. We argue that the results of our analysis are relevant not only to the particular task we have chosen, but to GA behavior in general.

2. CA Review and Terminology

A CA is spatial lattice of N cells, each of which is in one of k states at time t. Each cell follows the same simple rule for updating its state; the cell's state s at time $t+1$ depends on its own state and the states of some number of neighboring cells at time t. For one-dimensional CAs, the neighborhood of a cell consists of the cell itself and r neighbors on either side. The number of states k and the radius r are parameters of the CA.

The CA starts out with some initial configuration (IC) of cell states, and at each time step the states of all cells in the lattice are synchronously updated. We use the term "state" to refer to the local state s — the value of a single cell. Here we will restrict our attention to binary ($k = 2$) CAs with $s \in \{0, 1\}$. The state at site i is denoted by s^i. The term "configuration" will refer to the pattern of local states over the entire lattice. This is the CA's global state, denoted $\mathbf{s} = s^0 s^1 \ldots s^{N-1}$. The density of 1s in a configuration $\mathbf{s}$ will be denoted $\rho(\mathbf{s})$.

The equations of motion ϕ for a CA (the CA "rule") can be expressed as a look-up table that lists, for each local neighborhood, the update state for the neighborhood's central cell. A sample rule (the "majority" rule) for a one-dimensional "elementary" ($k = 2, r = 1$) CA is the following. Each possible neighborhood η is given along with the "output bit" $s = \phi(\eta)$ to which the central cell is updated.

η	000	001	010	011	100	101	110	111
s	0	0	0	1	0	1	1	1

In words, this rule says that for each neighborhood of three adjacent cells, the new state is decided by a majority vote among the three cells. At time step t, this look-up table is

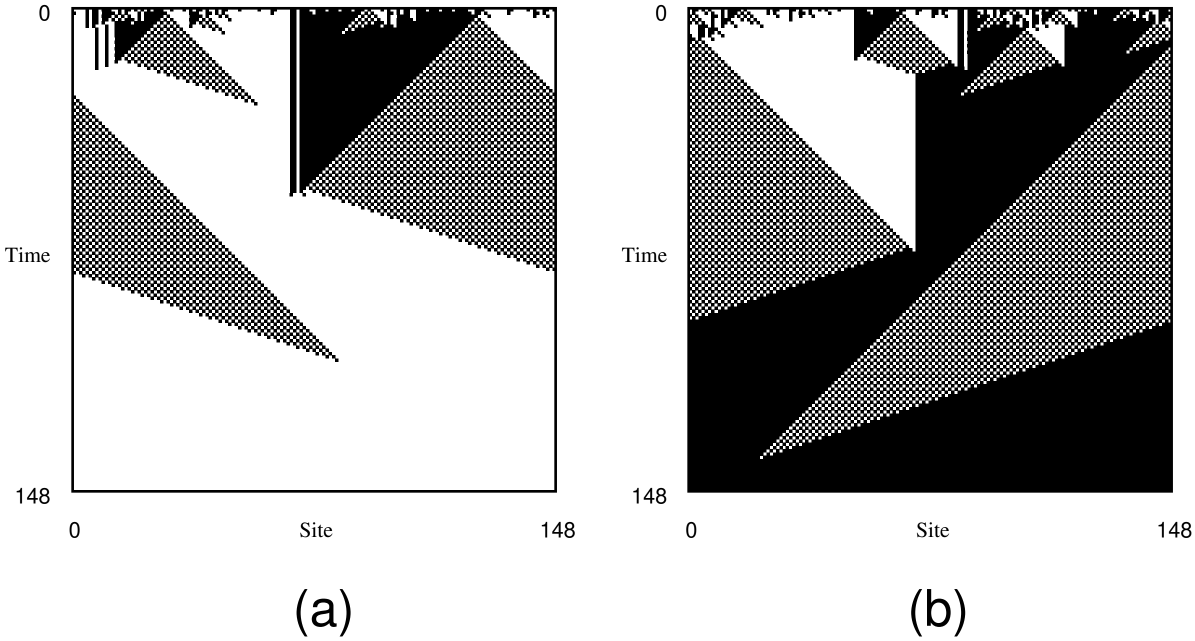

Figure 1: Two space-time diagrams for the binary-state Gacs-Kurdyumov-Levin CA. $N = 149$ sites are shown evolving, with time increasing down the page, from two different ICs over 149 time steps. Here cells with state 0 are white, and cells with state 1 are black. In (a), $\rho_0 \approx 0.48$, and in (b), $\rho_0 \approx 0.52$. Notice that by the last time step the CA has converged to a fixed pattern of (a) all 0s and (b) all 1s. In this way the CA has classified the ICs according to whether $\rho_0 > 1/2$ or $\rho_0 < 1/2$.

applied to each neighborhood in the current lattice configuration, respecting the choice of boundary conditions, to produce the configuration at $t + 1$. The configuration at time t will be denoted $\mathbf{s}_t = s_t^0 s_t^1 \ldots s_t^{N-1}$, where s_t^i is the local state of site i at time t. The CA equations of motion then specify a spatially-local update of a site's value as a function of its neighborhood: $s_{t+1}^i = \phi(\eta_t^i)$, where η_t^i is the neighborhood pattern about site i at time t. This local update induces a global mapping Φ that takes a lattice configuration at t to a new configuration at $t+1$: $\mathbf{s}_{t+1} = \Phi(\mathbf{s}_t)$. This can also be denoted in terms of the t^{th} iterate of Φ as: $\mathbf{s}_{t+1} = \Phi_{t+1}(\mathbf{s}_0)$.

The λ value of a binary CA is defined as the fraction of 1s in the output bits of its rule ϕ. For example, the λ value of the majority rule is 1/2. The λ parameter was originally used in studies of CA behavior [46], but, as we will show, it turns out to be useful in understanding the GA's behavior. (There is a simple interpretation of how λ is related to a CA's behavior. λ gives the density of 1s in the first iterate of a random initial configuration $\mathbf{s}_0$: $\rho(\mathbf{s}_1) = \lambda$.)

The behavior of a one-dimensional CA is often presented as a "space-time diagram", a plot of $\mathbf{s}_t$ over a range of time steps. Two examples are given in Figure 1. These show the actions of the Gacs-Kurdyumov-Levin (GKL) CA [29] on two random ICs; one with $\rho_0 > 1/2$ and the other with $\rho_0 < 1/2$. (Here and later on we use the shorthand ρ_0 for the density $\rho(\mathbf{s}_0)$ of an IC.) In both cases, the CA relaxes to a fixed pattern—in one case all 0s

($\rho(\mathbf{s}_\infty) = 0$) and in the other case all 1s ($\rho(\mathbf{s}_\infty) = 1$). The GKL CA will be discussed further below.

In this paper we restrict the discussion to one-dimensional CAs with $k = 2$ and $r = 3$, and with spatially periodic boundary conditions: $s_t^i = s_t^{i+N}$. We most often set N to 149, but also look at the behavior of CA on larger N (up to 999).

3. Previous Work

In [60] we reported results of evolving one-dimensional CAs to perform a particular density classification task: the "$\rho_c = 1/2$" task. This work was a re-examination of an experiment performed by Packard [65], meant to test the hypothesis that a GA evolving CA rules to perform a difficult computational task will tend to select rules close to conjectured phase transitions in rule space between ordered and chaotic behavior ("the edge of chaos"). In [65], the locations of these phase transitions were claimed to correspond to "critical" λ values, λ_c. In [65] the GA tended to select rules close to these critical values; these results were interpreted by Packard as supporting the "edge of chaos" hypothesis. As reported in [60], however, our similar experiments did not support this hypothesis. We also gave a theoretical argument that the $\rho_c = 1/2$ task requires rules with $\lambda = 1/2$ rather than the λ_c values given in [65]. We argued that the results reported in [65] were an artifact of the particular GA used there rather than due to any intrinsic computational advantage of rules with $\lambda = \lambda_c$, and concluded that to date there is no direct experimental evidence linking computational capability with λ in cellular automata. For a review of these issues and of relations among computation, dynamics, and cellular automata, see [58].

Although the results in [65] were not replicated in our experiments, we did observe several interesting phenomena in the GA's behavior. In [60] we qualitatively described the "epochs of innovation" in the GA's search for successful rules, as well as a breaking of the task's symmetries on the part of the GA. We interpreted the symmetry-breaking as impeding the GA's ability to find the best-performing rules. We also described the competing pressures of selection and "combinatorial drift". In this paper we analyze these phenomena in detail and explain the mechanisms underlying the GA's behavior and the impediments the GA encounters.

4. The Computational Task

The $\rho_c = 1/2$ task is defined as follows. If $\rho_0 < \rho_c$, then the CA is to relax, after a certain number M of time steps, to a fixed pattern of all 0s; otherwise, it is to relax to a fixed pattern of all 1s. The desired behavior is undefined at $\rho_0 = \rho_c$; this case will be precluded by using odd N. On a N site lattice then we have

$$\mathbf{T}_{\rho_c}(N, M) = \left\{ \begin{array}{ll} \Phi_M(\mathbf{s}_0) = 0^N & \text{if } \rho(\mathbf{s}_0) < \rho_c \\ \Phi_M(\mathbf{s}_0) = 1^N & \text{if } \rho(\mathbf{s}_0) > \rho_c \\ \text{undefined} & \text{if } \rho(\mathbf{s}_0) = \rho_c \end{array} \right\} \forall \mathbf{s}_0 \in \{0,1\}^N$$

In this notation the $\rho_c = 1/2$ task is denoted $\mathbf{T}_{1/2}$. This task is an example of "useful computation" in our characterization of the different types of computation in CAs [60]. That is, the global mapping Φ_M is interpreted as a program for performing a useful computation, the IC $\mathbf{s}_0$ is interpreted as the input to that program, and the CA runs for some specified number M of time steps or until it reaches one of a set of "goal" patterns, 0^N or 1^N. The final pattern is interpreted as the output.

The task $\mathbf{T}_{1/2}$ is interesting for a number of reasons. Density classification is closely related to a number of image-processing tasks, and studying simple versions of such tasks in one dimension will help in understanding how to use the GA to scale up to more complex two-dimensional tasks. In addition, the task is nontrivial for a small-radius ($r \ll N$) CA, since density is a global property of a configuration, whereas a small-radius CA relies only on local interactions. In other words, the task difficulty derives from the fact that a CA is specified by ϕ but the useful computation is effected by the global map Φ_M. In fact, the minimum amount of memory for $\mathbf{T}_{1/2}$ is proportional to $log(N)$, since the equivalent of a counter register is required to track the excess of 1s in a serial scan of the IC. In other words, the task requires computation which corresponds to the recognition of a non-regular language. Since the 1s can be distributed throughout the CA lattice, the CA must transfer information over large space-time distances ($\approx N$).

$\mathbf{T}_{1/2}$ possesses two symmetries. Denoting the task's global mapping of strings $\mathbf{s} \in \{0,1\}^N$ to classifications {LO, HI} by T, these are given as follows.

1. If an IC $\mathbf{s}_0$ is spatially reversed on the lattice, T gives the same classification. That is, $T(\mathbf{s}) = T(\mathcal{R}\mathbf{s})$, where the symmetry operator $\mathcal{R}$ reverses the order of the bits in $\mathbf{s}_0$.
2. If all the bits in $\mathbf{s}_0$ are flipped (i.e., 1s are exchanged with 0s and 0s with 1s), then T gives the opposite classification. That is, $T(\mathbf{s}_0) = \mathcal{F}T(\mathcal{F}\mathbf{s}_0)$, where the symmetry operator $\mathcal{F}$ flips the bits in $\mathbf{s}_0$. $\mathcal{F}$ is its own inverse.

Thus, there are several symmetries in $\mathbf{T}_{1/2}$ that we expect any high-performance candidate rule to respect—either locally (i.e., with respect to individual neighborhoods in the rule table ϕ) or globally (i.e., with respect to Φ, the global mapping), or both.

The second symmetry of $\mathbf{T}_{1/2}$ has an important consequence when interpreted with respect to ρ_0. Recall that on exactly half the possible ICs — the low density $\mathbf{s}_0$ — the desired behavior is to relax to a fixed point of all 0s, and on the other half — the high density $\mathbf{s}_0$ — the desired behavior is to relax to a fixed point of all 1s. This ρ_0 symmetry requires that any rule that performs this task has $\lambda = 1/2$. Suppose, for example, a rule that carries out the $\mathbf{T}_{1/2}$ task has $\lambda < 1/2$. This implies that for the majority of neighborhoods η, $\phi(\eta) = 0$. This, in turn, means that there will be some $\mathbf{s}_0$ with $\rho(\mathbf{s}_0) > \rho_c$ on which the action of the rule will decrease $\rho(\mathbf{s})$. This is the opposite of the desired action. If the rule acts to decrease $\rho(\mathbf{s})$, it risks producing an intermediate configuration $\mathbf{s}_{t'}$ with $\rho(\mathbf{s}_{t'}) < \rho_c$. This then would lead, under the original assumption that the rule carries out the task correctly, to a fixed point of all 0s, misclassifying $\mathbf{s}_0$. A similar argument holds in the other direction for $\lambda > 1/2$. This informal argument shows that a rule with $\lambda \neq 1/2$ will misclassify certain ICs. Generally, the further away the rule is from $\lambda = 1/2$, the larger the fraction of misclassified ICs.

5. The Strategy of a Hand-Designed CA

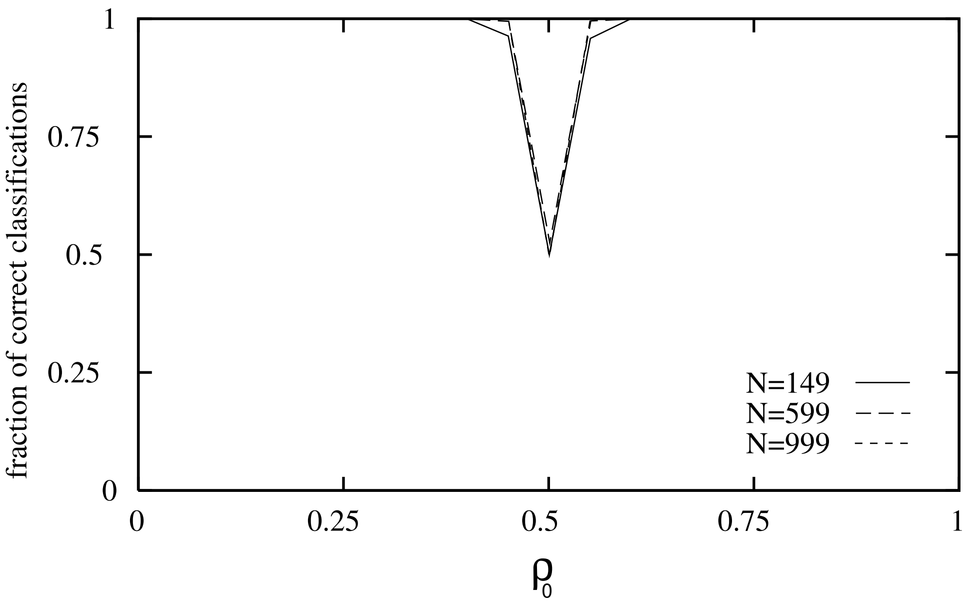

Figure 2: Experimental performance of the GKL rule as a function of ρ_0 for the $\rho_c = 1/2$ task. Performance plots are given for three lattice sizes: $N = 149$ (the size of the lattice used in the GA runs), 599, and 999. (This figure differs slightly from Figure 4 in [60], since 21 density bins were used there.) Note that the $N = 599$ and $N = 999$ curves are almost indistinguishable.

Does there exist a CA that can perform the $\rho_c = 1/2$ task? It is possible that no CA exists which performs the task perfectly for all N. However, a $k = 2, r = 3$ rule designed by Gacs, Kurdyumov, and Levin (the GKL rule) [29] appears to perform the task with error decreasing as $N \to \infty$ [47]. (We are not aware of any proof of this, however.) The observed classification performance of the GKL rule as a function of ρ_0 is given in Figure 2 for $N =$ 149, 599, and 999. To make this plot, we ran the GKL rule on 500 randomly generated ICs close to each of 19 densities $\rho \in [0.0, 1.0]$. The fraction of correct classifications was then plotted at each ρ_0. The rule was run either until a fixed point was reached or for a fixed maximum number of time steps $M = 10 \times N$.

Figure 2 indicates that all the misclassifications occur for $\rho_0 \approx \rho_c$, with the width of the error region decreasing as N increases. At $\rho_0 = \rho_c$, in fact, it appears no better than an unbiased random classification. We found that most errors were a result of relaxing to the wrong fixed point (e.g., all 0s for $\rho_0 > \rho_c$). For future reference note that on an $N = 149$ lattice the GKL rule's performance on $\rho_c = 1/2$ classification is ≈ 0.972 when averaged over 10^4 ICs uniformly distributed in 19 equally-spaced ρ bins.

The GKL rule is instructive for the density-classification task in that it happens to give a concrete, though approximate, solution to the optimization facing the GA. The manner in which it implements the required computation, its "strategy", is of equal importance. The rule's "strategy" here refers to the behavioral elements employed during its temporal evolution that effect the classification.

It should be emphasized, however, that the GKL rule was invented not for the purpose of performing any particular computational task, but rather as part of studies of reliable computation and phase transitions in one spatial dimension. The goal was to find a rule whose behavior is robust to small errors in the rule's update of the configuration. Reliable computation in this context meant the robust storage of a single bit of information in the presence of arbitrarily small noise. A zero or one was encoded as the CA configuration being close to an all 0s or all 1s pattern, respectively. In the absence of noise, it has been proved that the GKL rule has only two attracting patterns, either all 0s or all 1s [20]. Attracting patterns here are those invariant patterns which, when perturbed a small amount, return to the same pattern under the noise-free rule. Figure 2 shows that the basins of attraction for the all-1 and all-0 patterns are not precisely the ICs with $\rho_0 > 1/2$ or $\rho_0 < 1/2$, respectively. If they did coincide then the GKL rule would exactly implement $\rho_c = 1/2$-density classification.

The GKL rule is given by an equation of motion ϕ that updates the current configuration $\mathbf{s}_t = s_t^0, s_t^1, \ldots, s_t^{N-1}$ as follows

$$s_{t+1}^i = \phi(\eta_t^i) = \begin{cases} \text{majority}[s_t^i, s_t^{i-1}, s_t^{i-3}] & \text{if } s_t^i = 0 \\ \text{majority}[s_t^i, s_t^{i+1}, s_t^{i+3}] & \text{if } s_t^i = 1 \end{cases}$$

In words, this rule says that for each neighborhood η^i of seven adjacent cells, if the state of the central cell is 0, then its new state is decided by a majority vote among itself, its left neighbor, and the cell three sites to the left. Likewise, if the state of the central cell is 1, then its new state is decided by a majority vote among itself, its right neighbor, and the cell three sites to the right.

By listing the neighborhoods in lexicographic order, increasing from 0000000 to 1111111, the output bits of the GKL rule table can be given by a string $\mathbf{C}$ as follows.

$$\begin{aligned} \mathbf{C} \;=\; & 00000000010111110000000001011111 \\ & 00000000010111110000000001011111 \\ & 00000000010111111111111101011111 \\ & 00000000010111111111111101011111 \end{aligned} \qquad (5.1)$$

This lexicographic ordering is how CA rule tables will be represented as chromosomes to the GA. As expected, the GKL rule's λ value is exactly 1/2.

Locally in space-time the GKL dynamic ϕ does not satisfy the task symmetries individually. Over one time step it does so in a composite way: $\phi(\eta) = \mathcal{F} \circ \phi(\mathcal{F} \circ \mathcal{R}\eta)$. That is, if the neighborhood pattern is spatially-reversed and the bits are flipped, the opposite output bit results. This can be seen in Figure 1. Roughly speaking, inverting white to black and black to white and spatially reversing the patterns takes the downward pointing cross-hatched region in a black sea (Figure 1(b)) to the same in a white sea (Figure 1(a)). This composite symmetry is more stringent than that required by $\mathbf{T}_{1/2}$. Moreover, under the lexicographic ordering of neighborhoods, the composite symmetry imposes constraints on pairs of output bits that are spread throughout $\mathbf{C}$. The functionality of contiguous bits is a feature to which the GA's genetic operators can be sensitive.

Typical space-time behaviors of the GKL rule for ICs with $\rho_0 < \rho_c$ and $\rho_0 > \rho_c$ were shown in Figure 1. It can be seen that, although the patterns eventually converge to fixed points, there is a transient phase during which a spatial and temporal transfer of information

Particle	Wall Type	Velocity
a	WB	0
b	$\#W$	1
c	$W\#$	3
d	$B\#$	-1
e	$\#B$	-3
f	BW	0

Table 1: The six particles generated by the GKL CA.

about local regions takes place. This local information interacts with other local information to produce the desired final state. Very crudely, the GKL rule successively classifies "local" densities with a locality range that increases with time. In regions where there is some ambiguity, a "signal" is propagated. This is seen either as a checkerboard pattern propagated in both spatial directions or as a vertical white-to-black boundary. These signals indicate that the classification is to be made at a larger scale. Note that regions centered about each signal locally have $\rho = \rho_c$. The consequence is that the signal patterns can propagate, since the density of patterns with $\rho = \rho_c$ is neither increased nor decreased under the rule.

In this way, local information processing at later times classifies larger patches of the IC. In a simple sense, this summarizes the rule's "strategy" for performing the computational task. But how is the strategy related to the CA's dynamical behavior? The overall strategy can be decomposed into the CA's intrinsic computational elements: domains, particles, and particle interactions [36]. There are three time-invariant spatially-homogeneous domains: (i) all white, $W = 0^*$, (ii) all black, $B = 1^*$, and (iii) checkerboard, $\# = (10)^* \bigcup (01)^*$. The results in [20] establish that W and B are regular attractors, as defined in [36]. When the domains are filtered out using the methods of [16], one finds that the domain boundaries form six particles, the first five of which are time-invariant. These are listed in Table 1. The types of interactions between particles are also evident when the space-time diagrams are filtered as in [16]. There are two annihilative interactions: $c + b \rightarrow \emptyset$ and $d + e \rightarrow \emptyset$. Three of the interactions are reactive: $a + d \rightarrow c$, $b + a \rightarrow e$, and $c + e \rightarrow a$. There is one spontaneous decay: $f \rightarrow d + b$. At moderate to long times, it is these particle interactions which perform the local, but "emergent" logic that classifies successively larger portions of the IC. (A more complete analysis along these lines will be presented elsewhere.) As will be seen shortly, dynamical structures like these, and a few others, will be what the GA takes advantage of in evolving CA to implement the computational task.

6. Details of the GA and CAs in Our Experiments

Following Packard [65], we used a form of the GA to evolve one dimensional $k = 2$, $r = 3$ CAs to perform the $\rho_c = 1/2$ task. The k and r values were chosen to match those of the GKL rule. The GA begins with a population of P randomly generated rules: the "chromosomes", which are strings containing the rule table output bits. Like the bit-string listing of the GKL rule given above (equation 5.2), the output bits are given in lexicographic order of neighborhood patterns. For $k = 2$, $r = 3$ rules, the chromosomes representing rules are of length $2^{2r+1} = 128$. The size of the rule space the GA searches is thus 2^{128}—far too large for any kind of exhaustive search.

The fitness of a rule in the population is calculated by: (i) randomly choosing I ICs that are uniformly distributed over $\rho_0 \in [0.0, 1.0]$, with exactly half with $\rho_0 < \rho_c$ and half with $\rho_0 > \rho_c$; (ii) running the rule on each IC either until it arrives at a fixed point or for a maximum of M time steps; (iii) determining whether or not the final pattern is correct—i.e., $\mathbf{s}_M = 0^N$ with $\rho_0 < \rho_c$ and $\mathbf{s}_M = 1^N$ with $\rho_0 > \rho_c$. ρ_0 is never exactly 1/2, since N is chosen to be odd. The rule's fitness is the fraction of the I ICs on which the rule produces the correct final pattern.

This fitness function was termed "performance fitness" in [60]. It differs from "proportional fitness" in which the rule is given partial credit equal to the fraction of correct bits in the final pattern. The runs using performance fitness produced qualitatively similar results to those using proportional fitness [60], and in this paper we restrict our attention to the former. We denote the performance-fitness function using I ICs by F_I.

Our GA works as follows. At each generation:

1. A new set of I ICs is generated.
2. $F_I(\phi)$ is calculated for each rule ϕ in the population.
3. The population is ranked in order of fitness. (The ranking of rules with equal fitness is decided at random.)
4. A number E of the highest fitness ("elite") rules is copied without modification to the next generation.
5. The remaining $P - E$ rules for the next generation are formed by crossovers between randomly chosen pairs of elite rules. The parent rules are chosen from the elite with replacement. The offspring from each crossover are each mutated m times.

This defines one generation of the GA; it is repeated G times for one run of the GA. An experiment consists of a set of runs with identical parameters but different random number seeds.

Our experiments used single-point crossover, which takes two strings, selects a position at random, and forms two offspring by exchanging parts of the strings before and after that position. Mutation consists of flipping a randomly chosen bit in a string.

The fitness function F_I is an estimate of the true fitness F_{2^N}. It is a random variable, in fact, since the precise value it returns for a given rule depends on the particular set of I ICs used to test the rule. Thus a rule's fitness can vary stochastically from generation to generation. For this reason, at each generation the entire population, including the elite rules, is re-evaluated on a new set of ICs.

The parameter values in our main experiment were the following. (Subsequent sections will describe other experiments in which some parameter values were modified.) For each CA in the population: $N = 149$; $I = 100$, with ICs uniformly distributed over $\rho_0 \in [0.0, 1.0]$, half with $\rho_0 < \rho_c$ and half with $\rho_0 > \rho_c$; and $M \approx 320$. Each time a CA was simulated, M was chosen from a Poisson distribution with mean 320. This mean is the measured maximum amount of time for the GKL CA to reach an invariant pattern over a large number of ICs on lattice size 149. Varying M prevents overfitting of rules to a particular M; see [60]. In [60], I was set to 300, but we later found that setting I to 100 did not significantly change the results of our experiments and greatly reduced the required computation time. For the GA runs the chromosomes in the initial population were uniformly distributed over $\lambda \in [0.0, 1.0]$ and we set $P = 100$; $E = 20$; $m = 2$; and $G = 100$.

In GA parlance, our GA has a "generation gap"—the fraction of new strings in the next generation—of $1 - E/P = 0.8$. That is, once the population is ordered according to fitness, the top 20% of the population—the set of elite strings—is copied without modification into the next generation. Since testing a rule on 100 ICs provides only an approximate gauge of the true fitness, this relatively small generation gap was a good way of making a "first cut" and allowing rules that survive to be tested over more ICs. Since a new set of ICs was produced every generation, rules that were copied without modification were always retested on this new set. If a rule performed well and thus survived over a large number of generations, then it was likely to be a genuinely better rule than those that were not selected, since it was tested with a large set of ICs. An alternative method would be to test every rule in each generation on a much larger set of ICs, but this would waste computation time. Too much effort, for example, would go into testing very weak rules, which can safely be weeded out early using our method. As in most GA applications, in our GA the fitness-function evaluation dominates the required computational resources.

7. The GA's Epochs of Innovation

In [60] we qualitatively described a series of "epochs of innovation" that we observed in the GA runs using the proportional fitness function. We defined the "onset" of an epoch to be the generation at which a rule with a significant innovation in strategy was discovered. The onset generation of each epoch corresponded to a marked jump in the best fitness measured in the population. In this section we describe in more detail similar phenomena that we observed in a set of experiments using F_{100} that were performed subsequent to those reported in [60]. The account of the epochs given here differs slightly from—and is more rigorous than—that given in [60]. We will distinguish the onset generation from the "takeover" generation in which all or almost all of the elite rules implement the epoch's strategy.

We performed a total of 50 runs of the GA with the parameters given above. We also performed 50 runs of the GA with no crossover, 50 runs of the GA with no crossover and an initial population clustered close to $\lambda = 1/2$, and 22 runs of a Monte Carlo search method. Some statistics from these various runs are given in Table 2. Those given in columns 2–4 will be discussed in subsequent sections.

Figure 3 displays the best fitness at each generation for two typical GA runs (with crossover). The best fitnesses found under F_{100} ranged from 0.9–1.0. The standard deviation of F_{100}, when run 100 times on the same rule, is approximately 0.02. Naturally, it would be preferable to use a larger number of ICs to evaluate fitness during the evolution process, but this is computationally expensive. To obtain a truer value for best fitness after each run, we evaluated each of the best rules at the last generation of each run with 10^4 randomly chosen ICs, uniformly distributed over $\rho \in [0.0, 1.0]$. This fitness function is denoted F_{10^4}. Under F_{10^4}, all but one of the best fitnesses were between 0.883 and 0.945. The standard deviation

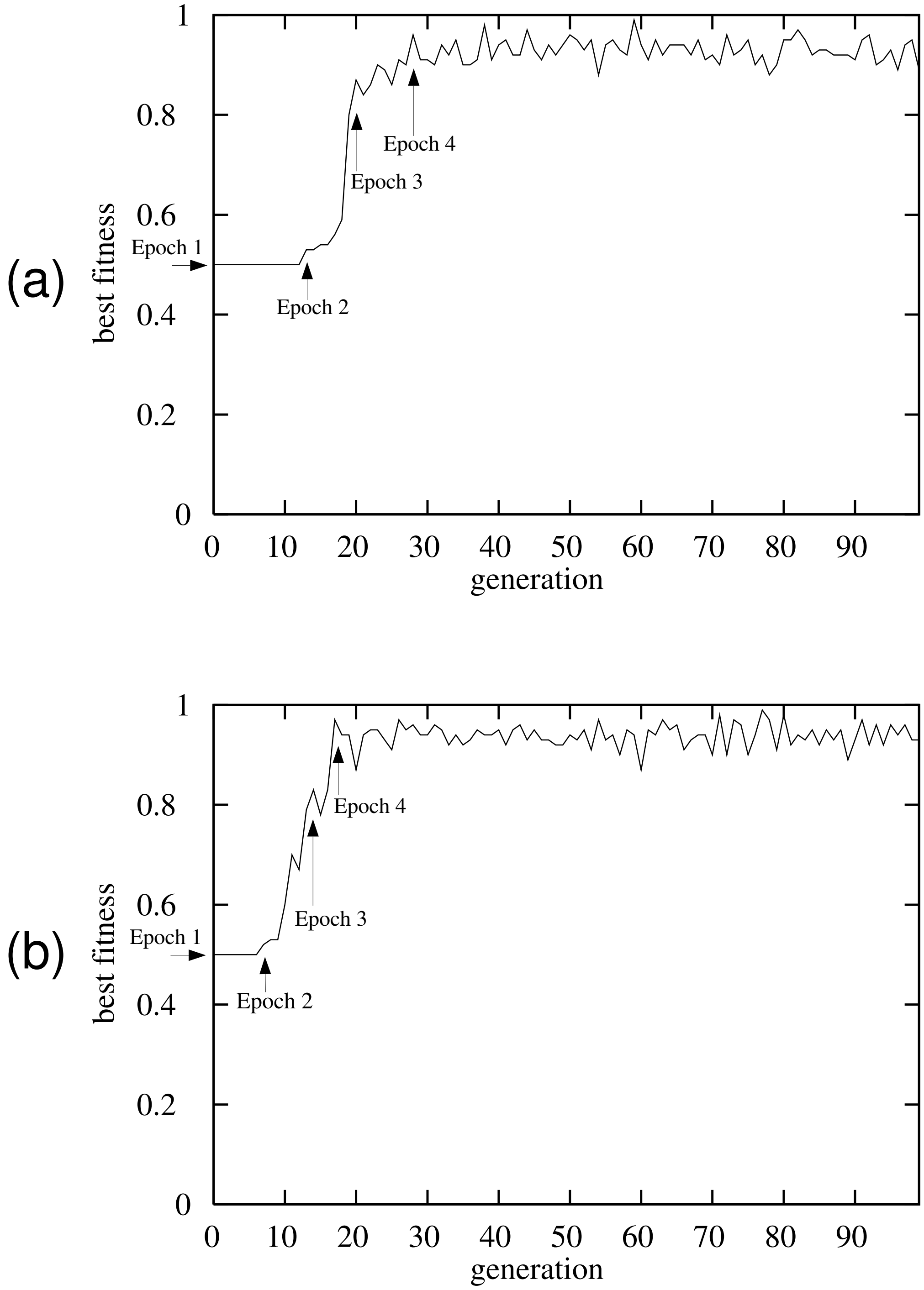

Figure 3: Best fitness versus generation for two typical runs. The onsets of four epochs of innovation—each corresponding to the generation discovery of a new, fitter strategy—are marked.

	GA, xover	GA, no xover	GA, no xover, initpop 1/2	Monte Carlo
Runs reaching Epoch 3	46/50 (92%)	13/50 (26%)	22/50 (44%)	8/22 (36%)
Runs used in averages	44/50	13/50	22/50	8/22
T_2	3.6 (3.3)	65.9 (12.9)	39.3 (20.6)	43.1 (18.7)
$T_3 - T_2$	3.8 (2.8)	9.9 (5.3)	7.6 (3.5)	4.5 (4.0)

Table 2: Fraction of runs reaching Epoch 3, fraction of runs used to compute averages (for "GA, xover" case, two outlyer runs were ommitted), mean generations to onset of Epoch 2 (T_2), and mean length of Epoch 2 in generations ($T_3 - T_2$) for those runs reaching Epoch 3 by generation 99 in four different experiments. Standard deviations are given in parentheses.

of F_{10^4}, when run 100 times on the same rule, is approximately 0.002. Recall that under F_{10^4}, the fitness of the GKL rule is 0.972. Under F_{10^4}, this is a significantly higher level of fitness than the level achieved by the GA. Therefore, the GA did not succeed in evolving the GKL rule, or any rule at a similar level of performance. Indeed, as we will discuss, the GA evolved a different set of strategies than ones that might be expected from the GKL rule.

The two plots in Figure 3 have four similar large-scale features, which turn out to correspond to four epochs of innovation. As was defined in [60], the onset of each epoch corresponds to the discovery of a new, higher-fitness strategy. The onset generation of each new epoch was determined by examining the actual strategies carried out by the elite rules in each run—i.e., the actual space-time dynamics of each rule. The onset generations are indicated in Figures 3. At generation 0, the onset of Epoch 1, the best fitness is 0.5 and remains at that level for several generations. Then there is a gradual or sharp rise in best fitness to $F_{100} \approx 0.53 - 0.70$ at to the onset of Epoch 2. This is followed by a sharp rise to 0.80 or higher at the onset of Epoch 3. The sharp rise is followed by a sharp or gradual rise to 0.9 or higher, corresponding to the onset of Epoch 4. The fitness then stays relatively constant, with moderate fluctuations arising from the stochastic nature of F_{100}. These same large-scale features are seen in the best-fitness time histories for almost every run. We examined rules in each run at different stages and, in almost all runs, observed roughly the same progressions through epochs and similar strategies at each epoch. Out of the 50 GA runs performed, 46 displayed a best-fitness plot similar to those in Figure 3. The only major differences were the generation of onset of Epoch 2 (T_2) and the length of Epoch 2 ($T_3 - T_2$, where T_3 is the generation of onset of Epoch 3). The mean values of T_2 and $T_3 - T_2$ over 44 of these runs are given in the first column of Table 2.

Two "outlier" runs were omitted from these averages. In these, $T_2 = 48, T_3 = 61$ and $T_2 = 49, T_3 = 58$ respectively. The best fitnesses in the remaining four runs never went beyond 0.5; we assume that if those runs had been allowed to continue, Epochs 2–4 eventually would have been reached.

The common strategies implemented by the best rules in different epochs are illustrated by the space-time diagrams in Figures 4–7. These are discussed in order below.

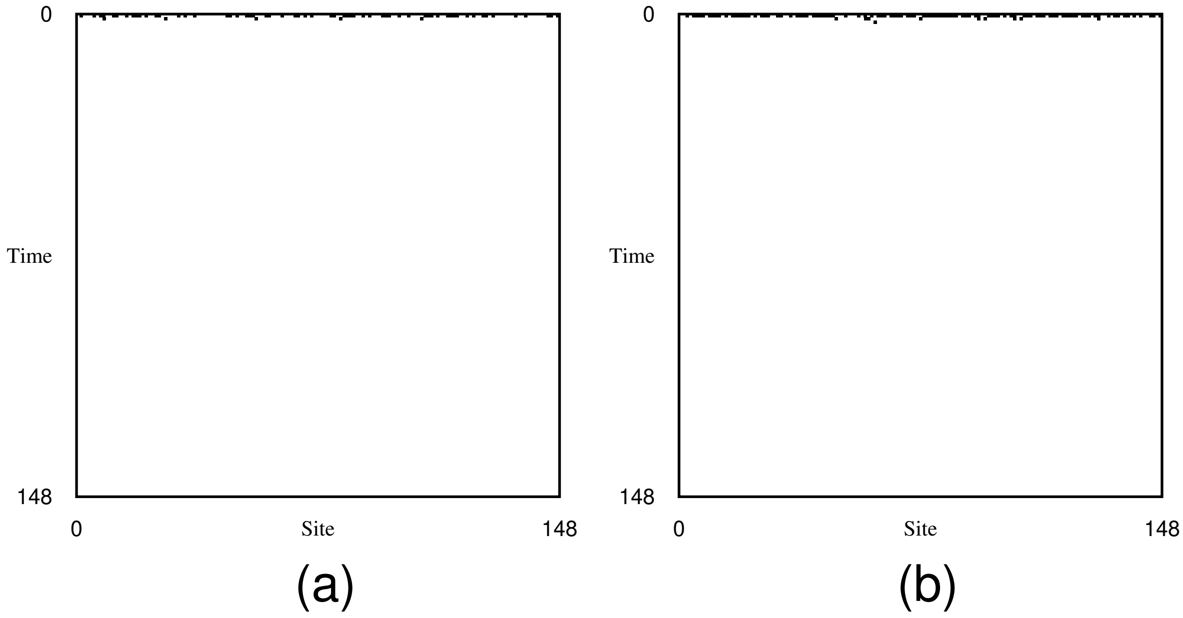

Figure 4: Two spacetime diagrams for an Epoch 1 rule with $\lambda \approx 0.05$.
In (a), $\rho_0 \approx 0.40$ and in (b), $\rho_0 \approx 0.67$.

Epoch 1: Best rules specialize on low ρ_0 or high ρ_0

In Epoch 1 there are two best-performing strategies: rules that always relax to a fixed point of all 0s and rules that always relax to a fixed point of all 1s. Figure 4 illustrates the former strategy on two ICs with low and high ρ_0, respectively. Since exactly half the ICs at each generation have $\rho_0 < \rho_c$ and exactly half have $\rho_0 > \rho_c$, each of these strategies is correct on exactly half the ICs, so each has fitness 0.5. This default behavior is hardly worthy of the name "strategy", but these rules perform significantly better than randomly chosen rules, which classify almost no ICs correctly. Since the rules in the initial populations are uniformly distributed over $\lambda \in [0.0, 1.0]$, the initial population always contains both very low and very high λ rules, which tend to have this behavior. This is why the best fitness in the initial population is almost always 0.5.

Epoch 2: Best rules correctly classify additional "extreme" ρ_0

At Epoch 2, the GA discovers rules that, while similar in behavior to Epoch 1 rules, correctly classify some additional ICs with extreme ρ_0. The behavior of one such rule is illustrated in Figure 5. Like the rule illustrated in Figure 4, this rule is a "low-ρ_0 specialist". But unlike the previous rule, it correctly classifies some very high ρ_0 ICs as well. In Figure 5(a), $\rho_0 < \rho_c$ and the CA quickly relaxes to all 0s. In Figure 5(b), $\rho_0 > \rho_c$ and the CA again relaxes to all 0s (a misclassification), but information from high-density blocks in the IC persist for some time. In Figure 5(c), $\rho_0 \gg \rho_c$ and the pattern is correctly classified. An Epoch 2 rule's additional correct classifications of very high (or very low) ρ ICs yields a slightly higher fitness, as seen in Figure 3. On approximately half the runs the GA discovers

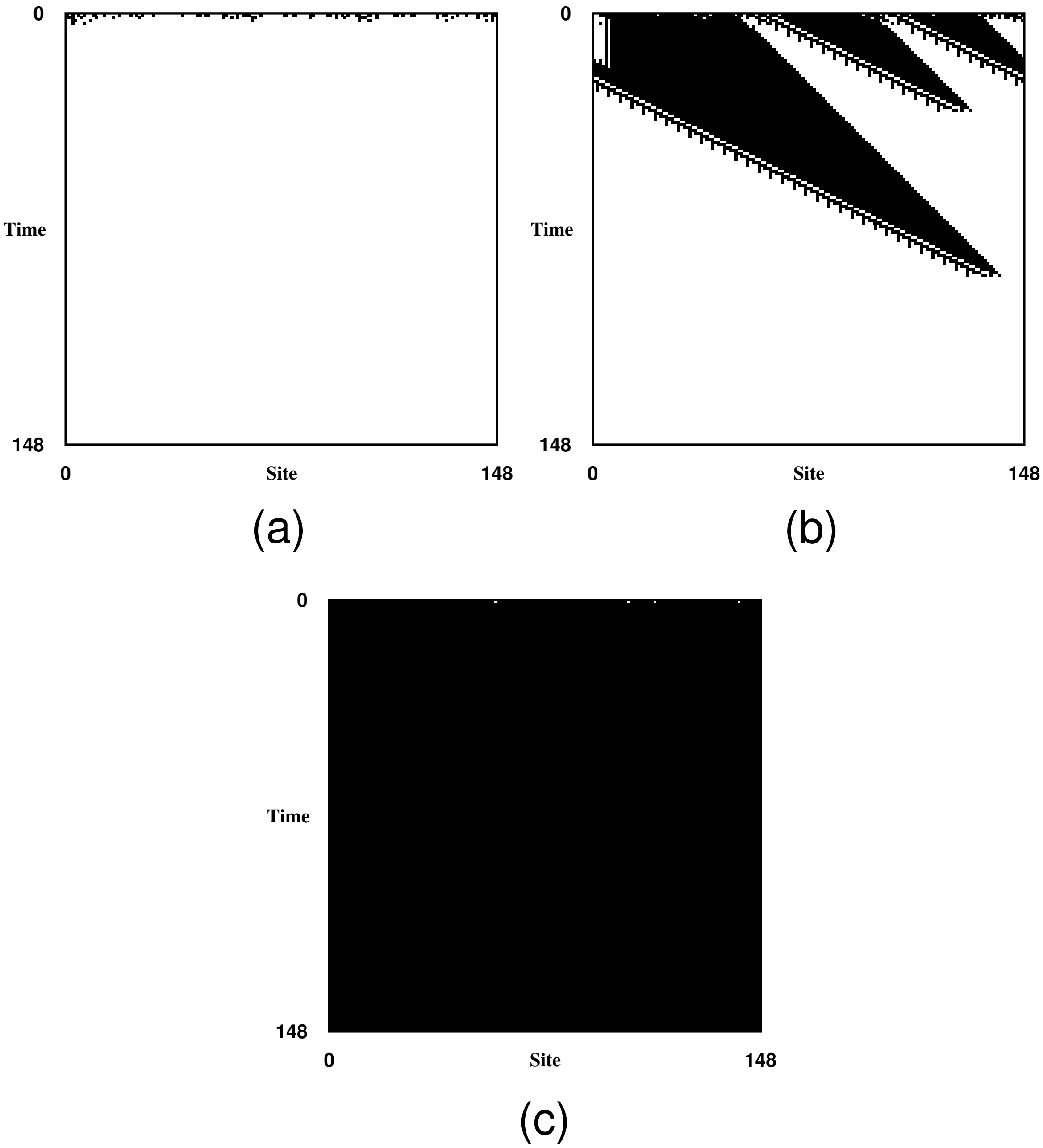

Figure 5: Three spacetime diagrams for an Epoch 2 rule with $\lambda \approx 0.33$. In (a), $\rho_0 \approx 0.37$, in (b), $\rho_0 \approx 0.86$, and in (c), $\rho_0 \approx 0.97$.

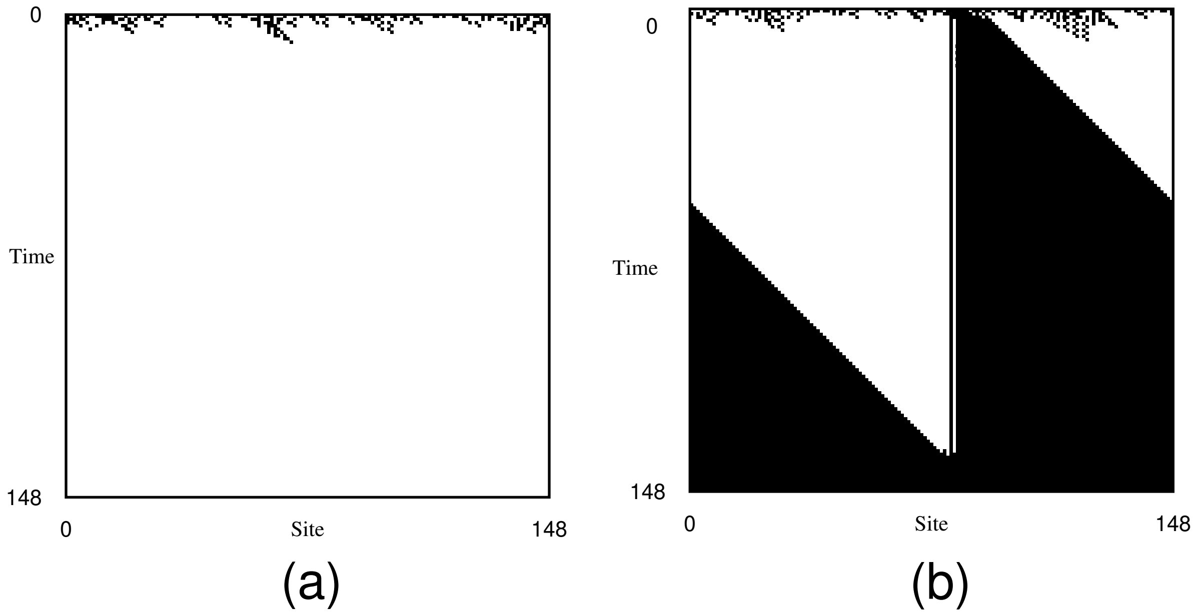

Figure 6: Two spacetime diagrams for an Epoch 4 rule with $\lambda \approx 0.38$ that implements Strategy 1. In (a), $\rho_0 \approx 0.41$ and in (b), $\rho_0 \approx 0.52$.

Epoch 2 low-ρ_0 specialists and on the other half it discovers Epoch 2 high-ρ_0 specialists. The strategies are almost never found together in the same run. Rules in Epoch 2 have fitnesses ranging from 0.51 to about 0.75, depending on how many additional high-density ICs are classified correctly and on the particular set of ICs being used at a given generation.

Epochs 3 and 4: Expanding blocks of 0s or 1s

Epoch 3 is characterized by a major innovation discovered by the GA. As in Epochs 1 and 2, there are two opposite strategies in Epoch 3:

- *Strategy 1:* Relax to a fixed point of all 0s unless there is a sufficiently large block of adjacent (or almost adjacent) 1s in the IC. If so, expand that block.
- *Strategy 2:* Relax to a fixed point of all 1s unless there is a sufficiently large block of adjacent (or almost adjacent) 0s in the IC. If so, expand that block.

The meaning of "sufficiently large" varies from rule to rule, and generally ranges from around 6 to 11 cells. (Note that "sufficiently large" can be larger than the neighborhood size $2r+1 = 7$. This can occur via the interaction between adjacent neighborhoods on the lattice.) As will be seen, in the higher-fitness rules, the size of blocks that are expanded is tuned to be a good predictor of high or low density for $N = 149$.

Epoch 3 begins with the discovery of such a rule, which typically has fitness $F_{100} \approx 0.8$. During Epoch 3, the GA discovers variants on the original rule and small improvements to these rules, having the effect of raising their fitnesses to $F_{100} \approx 0.9$. Epoch 4 begins

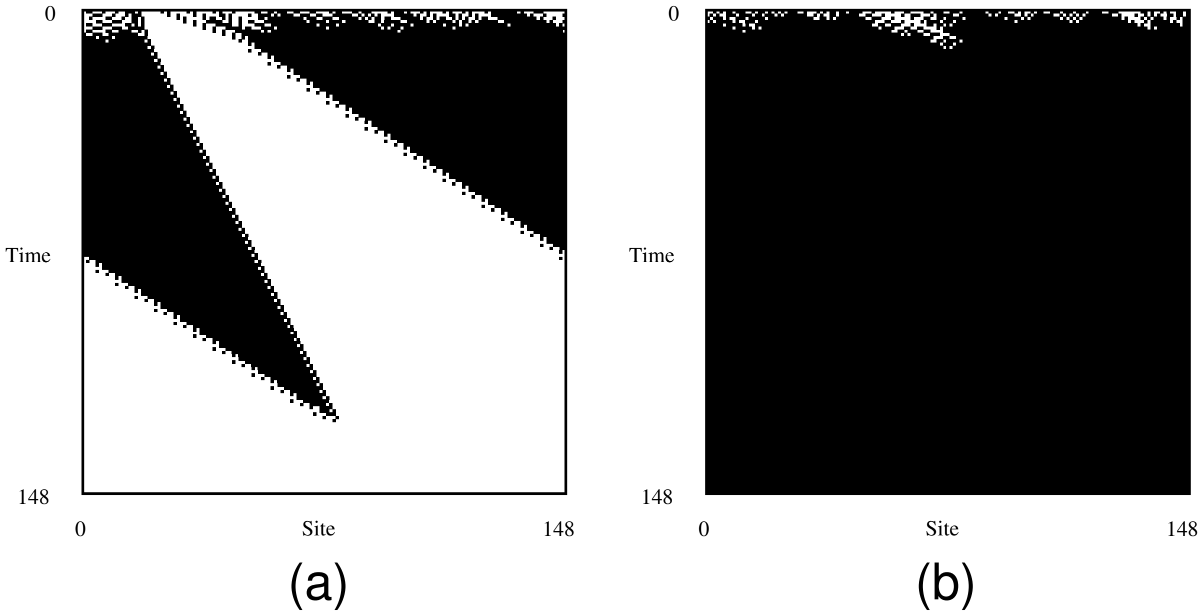

Figure 7: Two spacetime diagrams for an Epoch 4 rule with $\lambda \approx 0.59$ that implements Strategy 2. In (a), $\rho_0 \approx 0.39$ and in (b), $\rho_0 \approx 0.54$.

when no additional improvements are made. From that time on, the best fitnesses remain approximately constant, though there is moderately high variation in F_{100} as seen in Figure 3. Two examples of such rules from Epoch 4 are given in Figures 6 and 7. The rule in Figure 6 implements Strategy 1. In 6(a), $\rho_0 < \rho_c$, and the CA quickly relaxes to all 0s. In 6(b), $\rho_0 > \rho_c$, and there is a sufficiently large block of 1s, which the CA expands toward the right until the configuration reaches a fixed point of all 1s. Both ICs are correctly classified. Figure 7 displays a rule with Strategy 2. In 7(a), $\rho_0 < \rho_c$, and there is a sufficiently large block of 0s which is expanded. In 7(b), $\rho_0 > \rho_c$, and the configuration quickly relaxes to all 1s. Again, both ICs are correctly classified.

The best rules in Epochs 3 and 4 are specialists for low or high ρ_0, but rather than ignoring the opposite half of the ICs as in Epoch 1 or only dealing with special extreme cases as in Epoch 2, these rules deal with the other half of the ICs by expanding sufficiently large blocks of the non-default state. In this way they obtain a marked increase in fitness. In effect, the strategy of Epochs 3 and 4 use the presence or absence of such blocks as local predictors of the global ρ_0.

Typical errors made in Epoch 3 are illustrated in Figure 8. In 8(a), an Epoch 3 rule expands blocks that are too small, resulting in an incorrect classification for $\rho_0 < \rho_c$. In 8(b), another Epoch 3 rule expands blocks too slowly from an IC with $\rho_0 > \rho_c$, eventually reaching a fixed point of all 1s but not by the maximum allotted number M of iterations. (In the figure, $M = 149$; in our experiments, $M \approx 320$.) This also results in a failure to correctly classify within the given time. In Figure 8(c), a third Epoch 3 rule *creates* a block not present in $\mathbf{s}$ and expands it, resulting in a misclassification for $\rho_0 \ll \rho_c$. Such errors are largely corrected by Epoch 4, though even the best Epoch 4 rules still misclassify a number