Revisiting the Edge of Chaos: Evolving Cellular Automata to Perform Computations

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

Abstract

We present results from an experiment similar to one performed by Packard [24], in which a genetic algorithm is used to evolve cellular automata (CA) to perform a particular computational task. Packard examined the frequency of evolved CA rules as a function of Langton's λ parameter [17], and interpreted the results of his experiment as giving evidence for the following two hypotheses: (1) CA rules able to perform complex computations are most likely to be found near "critical" λ values, which have been claimed to correlate with a phase transition between ordered and chaotic behavioral regimes for CA; (2) When CA rules are evolved to perform a complex computation, evolution will tend to select rules with λ values close to the critical values. Our experiment produced very different results, and we suggest that the interpretation of the original results is not correct. We also review and discuss issues related to λ, dynamical-behavior classes, and computation in CA.

The main constructive results of our study are identifying the emergence and competition of computational strategies and analyzing the central role of symmetries in an evolutionary system. In particular, we demonstrate how symmetry breaking can impede the evolution toward higher computational capability.

[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

The notion of "computation at the edge of chaos" has gained considerable attention in the study of complex systems and artificial life (e.g., [4, 5, 15, 17, 24, 31]). This notion is related to the broad question, What is the relationship between a computational system's ability for complex information processing and other measures of the system's behavior? In particular, does the ability for nontrivial computation require a system's dynamical behavior to be "near a transition to chaos"? There has also been considerable attention given to the notion of "the edge of chaos" in the context of evolution. In particular, it has been hypothesized that when biological systems must perform complex computation in order to survive, the process of evolution under natural selection tends to select such systems near a phase transition from ordered to chaotic behavior [14, 15, 24].

This paper describes a re-examination of one study that addressed these questions in the context of cellular automata [24]. The results of the original study were interpreted as evidence that an evolutionary process in which cellular-automata rules are selected to perform a nontrivial computation preferentially selected rules near transitions to chaos. We show that this conclusion is neither supported by our experimental results nor consistent with basic mathematical properties of the computation being evolved. In the process of this demonstration, we review and clarify notions relating to terms such as "computation", "dynamical behavior", and "edge of chaos" in the context of cellular automata.

2. Cellular Automata and Dynamics

Cellular automata (CA) are discrete spatially-extended dynamical systems that have been studied extensively as models of physical processes and as computational devices [7, 11, 26, 30, 32]. In its simplest form, a CA consists of a spatial lattice of *cells*, each of which, at time t, can be in one of k states. We denote the lattice size or number of cells as N. A CA has a single fixed rule used to update each cell; the rule maps from the states in a neighborhood of cells—e.g., the states of a cell and its nearest neighbors—to a single state, which is the update value for the cell in question. The lattice starts out with some initial configuration of local states and, at each time step, the states of all cells in the lattice are synchronously updated. In the following we will use the term "state" to refer to the value of a single cell—e.g., 0 or 1—and "configuration" to mean the pattern of states over the entire lattice.

The CA we will discuss in this paper are all one-dimensional with two possible states per cell (0 and 1). In a one-dimensional CA, the neighborhood of a cell includes the cell itself and some number of neighbors on either side of the cell. The number of neighbors on either side of the center cell is referred to as the CA's *radius* r. All of the simulations will be of CA with spatially periodic boundary conditions (i.e., the one-dimensional lattice is viewed as a circle, with the right neighbor of the rightmost cell being the leftmost cell, and vice versa).

The equations of motion for a CA are often expressed in the form of a *rule table*. This is a look-up table listing each of the neighborhood patterns and the state to which the central cell in that neighborhood is mapped. For example, the following displays one possible rule table for an "elementary" one-dimensional two-state CA with radius $r = 1$. Each possible neighborhood η is given along with the "output bit" $s = \phi(\eta)$ to which the central cell is

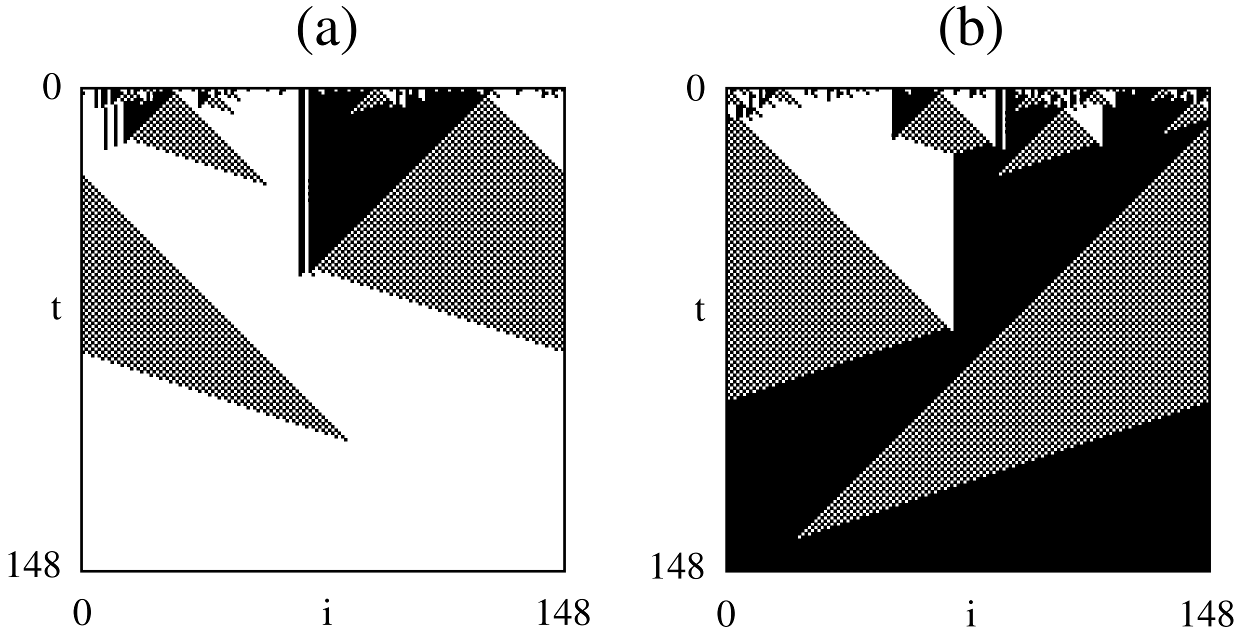


Figure 1: Two space-time diagrams for the binary-state Gacs-Kurdyumov-Levin CA. $N = 149$ sites are shown evolving, with time increasing down the page, from two different initial configurations over 149 time steps. In (a) the initial configuration has a density of 1's of approximately 0.48; in (b) a density of approximately 0.52. Notice that by the last time step the CA has converged to a fixed pattern of (a) all 0's and (b) all 1's. In this way the CA has classified the initial configurations according to their density.

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. To run the CA, this look-up table is applied to each neighborhood in the current lattice configuration, respecting the choice of boundary conditions, to produce the configuration at the next time step.

A common method for examining the behavior of a two-state one-dimensional CA is to display its space-time diagram, a two-dimensional picture that vertically strings together the one-dimensional CA lattice configurations at each successive time step, with white squares corresponding to cells in state 0, and black squares corresponding to cells in state 1. Two such space-time diagrams are displayed in Figure 1. These show the actions of the Gacs-Kurdyumov-Levin (GKL) binary-state CA on two random initial configurations of different densities of 1's [6, 8]. In both cases, over time the CA relaxes to a fixed pattern—in one case, all 0's, and in the other case, all 1's. These patterns are, in fact, fixed points of the GKL CA. That is, once reached, further applications of the CA do not change the pattern. The GKL CA will be discussed further below.

CA are of interest as models of physical processes because, like many physical systems, they consist of a large number of simple components (cells) which are modified only by local interactions, but which acting together can produce global complex behavior. Like the class of dissipative dynamical systems, even the class of elementary one-dimensional CA exhibit the full spectrum of dynamical behavior: from fixed points, as seen in Figure 1, to limit cycles (periodic behavior) to unpredictable ("chaotic") behavior. Wolfram considered a coarse classification of CA behavior in terms of these categories. He proposed the following four classes with the intention of capturing all possible CA behavior [31]:

Class 1: Almost all initial configurations relax after a transient period to the same fixed configuration (e.g., all 1's).

Class 2: Almost all initial configurations relax after a transient period to some fixed point or some temporally periodic cycle of configurations, but which one depends on the initial configuration.

Class 3: Almost all initial configurations relax after a transient period to chaotic behavior. (The term "chaotic" here and in the rest of this paper refers to apparently unpredictable space-time behavior.)

Class 4: Some initial configurations result in complex localized structures, sometimes long-lived.

Wolfram does not state the requirements for membership in Class 4 any more precisely than is given above. Thus, unlike the categories derived from dynamical systems theory, Class 4 is not rigorously defined.

It should be pointed out that on finite lattices, there is only a finite number (2^N) of possible configurations, so all rules ultimately lead to periodic behavior. Class 2 refers not to this type of periodic behavior but rather to cycles with periods much shorter than 2^N.

3. Cellular Automata and Computation

CA are also of interest as computational devices, both as theoretical tools and as practical highly efficient parallel machines [26, 27, 30, 32].

"Computation" in the context of CA has several possible meanings. The most common meaning is that the CA does some "useful" computational task. Here, the rule is interpreted as the "program", the initial configuration is interpreted as the "input", and the CA runs for some specified number of time steps or until it reaches some "goal" pattern—possibly a fixed point pattern. The final pattern is interpreted as the "output". An example of this is using CA to perform image-processing tasks [27].

A second meaning of computation in CA is for a CA, given certain special initial configurations, to be capable of universal computation. That is, the CA can, given the right initial configuration, simulate a programmable computer, complete with logical gates, timing devices, and so on. Conway's Game of Life [1] is such a CA; one construction for universal computation in the Game of Life is given in [1]. Similar constructions have been made for one-dimensional CA [21]. Wolfram speculated that all Class 4 rules have the capacity for universal computation [31]. However, given the informality of the definition of Class 4, not to mention the difficulty of proving that a given rule is or is not capable of universal computation, this hypothesis is impossible to verify.

A third meaning of computation in CA involves interpreting the behavior of a given CA on an ensemble of initial configurations as a kind of "intrinsic" computation. Here computation is not interpreted as the performance of a "useful" transformation of the input to produce the output. Rather, it is measured in terms of generic, structural computational elements such as memory, information production, information transfer, logical operations, and so on. It is important to emphasize that the measurement of such intrinsic computational elements does not rely on a semantics of utility as do the preceding computation types. That is, these elements can be detected and quantified without reference to any specific "useful" computation performed by the CA—such as enhancing edges in an image or computing the digits of π. This notion of intrinsic computation is central to the work of Crutchfield, Hanson, and Young [4, 12].

Generally, CA have both the capacity for all kinds of dynamical behaviors and the capacity for all kinds of computational behaviors. For these reasons, in addition to the computational ease of simulating them, CA have been considered a good class of models to use in studying how dynamical behavior and computational ability are related. Similar questions have also been addressed in the context of other dynamical systems, including continuous-state dynamical systems such as iterated maps and differential equations [4, 5], Boolean networks [14], and recurrent neural networks [25]. Here we will confine our discussion to CA.

With this background, we can now rephrase the broad questions presented in Section 1 in the context of CA:

- What properties must a CA have for nontrivial computation?
- In particular, does a capacity for nontrivial computation, in any of the three senses described above, require CA in a region of rule space near a transition from ordered to chaotic behavior?
- When CA rules are evolved to perform a nontrivial computation, will evolution tend to select rules near such a transition to chaos?

4. Structure of CA Rule Space

Over the last decade there have been a number of studies addressing the first question above. Here we focus on Langton's empirical investigations of the second question in terms of the structure of the space of CA rules [17]. The relationship of the first two questions to the third—evolving CA—will be described subsequently.

One of the major difficulties in understanding the structure of the space of CA rules and its relation to computational capability is its discrete nature. In contrast to the well-developed theory of bifurcations for continuous-state dynamical systems [10], there appears to be little or no geometry in CA space and there is no notion of smoothly changing one CA to get another "nearby in behavior". In an attempt to emulate this, however, Langton defined a parameter λ that varies incrementally as single output bits are turned on or off in a given rule table. For a given CA rule table, λ is computed as follows. For a k-state CA, one state q is chosen arbitrarily to be "quiescent".[3] The λ of a given CA rule is then the fraction of non-quiescent output states in the rule table. For a binary-state CA, if 0 is chosen to be the quiescent state, then λ is simply the fraction of output 1 bits in the rule table. Typically there are many CA rules with a given λ value. For a binary CA, the number is strongly peaked at $\lambda = 1/2$, due to the combinatorial dependence on the radius r and the number of states k. It is also symmetric about $\lambda = 1/2$, due to the symmetry of exchanging 0's and 1's. Generally, as λ is increased from 0 to $[1 - 1/k]$, the CA move from having the most homogeneous rule tables to having the most heterogeneous.

Langton performed a range of Monte Carlo samples of two-dimensional CA in an attempt to characterize their average behavior as a function of λ [17]. The notion of "average behavior" was intended to capture the most likely behavior observed with a randomly chosen initial configuration for CA randomly selected in a fixed-λ subspace. The observation was that as λ is incremented from 0 to $[1 - 1/k]$ the average behavior of rules passes through the following regimes:

$$\text{fixed point} \Rightarrow \text{periodic} \Rightarrow \text{“complex”} \Rightarrow \text{chaotic.}$$

That is, according to Figure 16 in [17], for example, the average behavior at low λ is for a rule to relax to a fixed point after a relatively short transient phase. As λ is increased, rules tend to relax to periodic patterns, again after a relatively short transient phase. As λ reaches a "critical value" λ_c, rules tend to have longer and longer transient phases. Additionally, the behavior in this regime exhibits long-lived, "complex"—non-periodic, but non-random—patterns. As λ is increased further, the average transient length decreases, and rules tend to relax to apparently random space-time patterns. The actual value of λ_c depends on r, k and the actual path of CA found as λ is incremented.

These four behavioral regimes roughly correspond to Wolfram's four classes. Langton's claim is that, as λ is increased from 0 to $[1 - 1/k]$, the classes are passed through in the order 1, 2, 4, 3. He notes that as λ is increased, "...one observes a *phase transition* between highly *ordered* and highly *disordered* dynamics, analogous to the phase transition between the *solid* and *fluid* states of matter." ([17], p. 13.)

According to Langton, as λ is increased from $[1 - 1/k]$ to 1, the four regimes occur in the reverse order, subject to some constraints for $k > 2$ [17]. For two-state CA, since behavior is necessarily symmetric about $\lambda = 1/2$, there are two values of λ_c at which the complex regime occurs.

How is λ_c determined? Following standard practice Langton used various statistics such as single-site entropy, two-site mutual information, and transient length to classify CA behavior. The additional step was to correlate behavior with λ via these statistics. Langton's Monte Carlo samples showed there was some correlation between the statistics and λ. But the averaged statistics did not reveal a sharp transition in average behavior, a basic property of a phase transition in which macroscopic highly-averaged quantities do make marked changes. We note that Wootters and Langton [33] gave evidence that in the limit of an increasing number of states the transition region narrows. Their main result indicates that

[3] In [17] all states obeyed a "strong quiescence" requirement. For any state $s \in \{0, ..., k-1\}$, the neighborhood consisting entirely of state s must map to s.

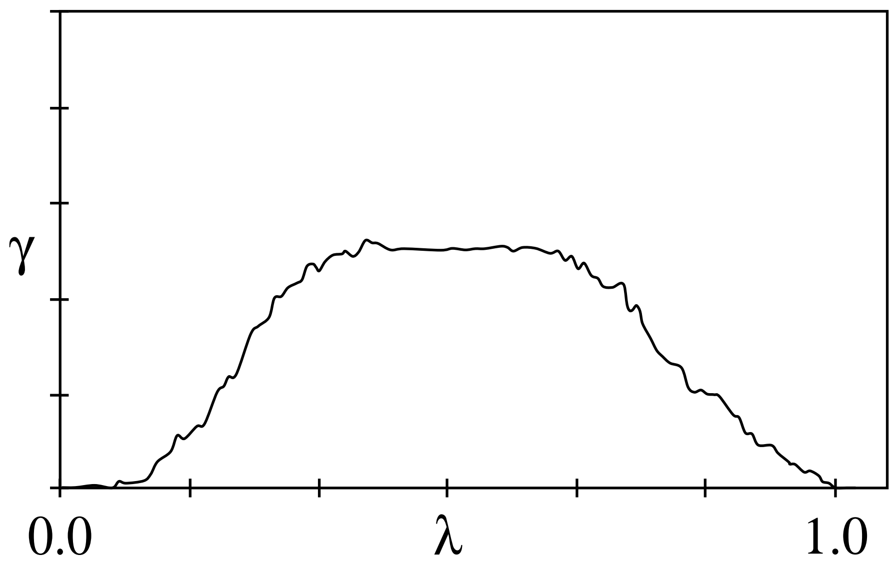


Figure 2: A graph of the average difference-pattern spreading rate γ of a large number of randomly chosen $r = 3, k = 2$ CA, as a function of λ. Adapted from [24], with permission of the author. No vertical scale was provided there.

in one class of two-dimensional infinite-state stochastic cellular automata there is a sharp transition in single-site entropy at $\lambda_c \approx 0.27$.

The existence of a critical λ and the dependence of the critical region's width on r and k is less clear for finite-state CA. Nonetheless, Packard empirically determined rough values of λ_c for $r = 3, k = 2$ CA by looking at the *difference-pattern spreading rate* γ as a function of λ [24]. The spreading rate γ is a measure of unpredictability in spatio-temporal patterns and so is one possible measure of chaotic behavior [22, 31]. It is analogous to, but not the same as, the Lyapunov exponent for continuous-state dynamical systems. In the case of CA it indicates the average propagation speed of information through space-time, though not the rate of production of local information.

At each λ a large number of rules was sampled and for each CA γ was estimated. The average γ over the selected CA was taken as the average spreading rate at the given λ. The results are reproduced in Figure 2. As can be seen, at low and high λ's, γ vanishes; at intermediate λ it is maximal, and in the "critical" λ regions—centered about $\lambda \approx 0.23$ and $\lambda \approx 0.83$—it rises or falls gradually.[4]

While not shown in Figure 2, for most λ values γ's variance is high. The same is true for single-site entropy and two-site mutual information as a function of λ [17]. That is, the behavior of any *particular* rule at a given λ might be very different from the *average* behavior at that value. Thus, the interpretations of these averages is somewhat problematic. The recounting given above of the behavioral structure of CA rule space as parameterized by λ is based on statistics taken from Langton's and Packard's Monte Carlo simulations. Various problems in correlating λ with behavior will be discussed in Section 8. A detailed analysis of some of these problems can be found in [3]. Other work on investigating the

[4]Li et al. (cf. [20], Appendix B) define λ_c as the onset of non-zero γ, and use mean-field theory to estimate λ_c in terms of r for two-state CA. The value from their formula, setting $r = 3$ is $\lambda_c = 0.146$, which roughly matches the value for the onset of non-zero γ seen in Figure 2.

structure of CA rule space is reported in [19, 20].

The claim in [17] is that λ predicts dynamical behavior well only when the space of rules is large enough. Apparently, λ is not intended to be a good behavioral predictor for the space of elementary CA rules—$r = 1$, $k = 2$—and possibly $r = 3, k = 2$ rules as well.

5. CA Rule Space and Computation

Langton [17] hypothesizes that a CA's computational capability is related to its average dynamical behavior, which λ is claimed to predict. In particular, he hypothesizes that CA capable of performing nontrivial computation—including universal computation—are most likely to be found in the vicinity of "phase transitions" between order and chaos, that is, near λ_c values. The hypothesis relies on a basic observation of computation theory, that any form of computation requires memory—information storage—and communication—information transmission and interaction between stored and transmitted information. Above and beyond these properties, though, universal computation requires memory and communication over arbitrary distances in time and space. Thus complex computation requires significantly long transients and space-time correlation lengths; in the case of universal computation, arbitrarily long transients and correlations are required. Langton's claim is that these phenomena are most likely to be seen near λ_c values—near "phase transitions" between order and chaos. This intuition is behind Langton's notion of "computation at the edge of chaos" for CA.[5]

6. Evolving CA

The empirical studies described above addressed only the relationship between λ and the dynamical behavior of CA—as revealed by several statistics. Those studies did not correlate λ or behavior with an independent measure of computation. Packard [24] addressed this issue by using a genetic algorithm (GA) [9, 13] to evolve CA rules to perform a particular computation. This experiment was meant to test two hypotheses: (1) CA rules able to perform complex computations are most likely to be found near λ_c values; and (2) When CA rules are evolved to perform a complex computation, evolution will tend to select rules near λ_c values.

6.1 The Computational Task and an Example CA

The original experiment consisted of evolving two-state—$s \in \{0, 1\}$—one-dimensional CA with $r = 3$. That is, the neighborhood of a cell consists of itself and its three neighbors on each side. The computational task for the CA is to decide whether or not the initial configuration contains more than half 1's. If the initial configuration contains more than half 1's, the desired behavior is for the CA, after some number of time steps, to relax to a fixed-point pattern of all 1's. If the initial configuration contains less than half 1's, the desired behavior is for the CA, after some number of time steps, to relax to a fixed-point

[5] This should be contrasted with the analysis of computation at the onset of chaos in [4, 5] and, in particular, with the discussion of the structure of CA space there.

pattern of all 0's. If the initial configuration contains exactly half 1's, then the desired behavior is undefined. This can be avoided in practice by requiring the CA lattice to be of odd length. Thus the desired CA has only two invariant patterns, either all 1's or all 0's. In the following we will denote the density of 1's in a lattice configuration by ρ, the density of 1's in the configuration at time t by $\rho(t)$, and the threshold density for classification by ρ_c.

Does the $\rho_c = 1/2$ classification task count as a "nontrivial" computation for a small-radius ($r \ll N$) CA? Though this term was not rigorously defined in [17] or [24], one possible definition might be any computation for which the memory requirement increases with N (i.e., any computation which corresponds to the recognition of a non-regular language) and in which information must be transmitted over significant space-time distances (on the order of N). Under this definition the $\rho_c = 1/2$ classification task can be thought of as a nontrivial computation for a small radius CA. The effective minimum amount of memory is proportional to $log(N)$ since the equivalent of a counter register is required to track the excess of 1's in a serial scan of the initial pattern. And since the 1's can be distributed throughout the lattice, information transfer over long space-time distances must occur. This is supported in a CA by the non-local interactions among many different neighborhoods after some period of time.

Packard cited a $k = 2, r = 3$ rule constructed by Gacs, Kurdyumov, and Levin [6, 8], which purportedly performs this task. The Gacs-Kurdyumov-Levin (GKL) CA is defined by the following rule:

$$\text{If } s_i(t) = 0, \text{ then } s_i(t+1) = \text{majority } [s_i(t),\, s_{i-1}(t),\, s_{i-3}(t)];$$
$$\text{If } s_i(t) = 1, \text{ then } s_i(t+1) = \text{majority } [s_i(t),\, s_{i+1}(t),\, s_{i+3}(t)];$$

where $s_i(t)$ is the state of site i at time t.

In words, this rule says that for each neighborhood 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 two cells to the left away. 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 two cells to the right away.

Figure 1 gives space-time diagrams for the action of the GKL rule on an initial configuration with $\rho < \rho_c$ and on an initial configuration with $\rho > \rho_c$. It can be seen that, although the CA eventually converges to a fixed point, there is a transient phase during which a spatial and temporal transfer of information about local neighborhoods takes place, and this local information interacts with other local information to produce the desired final state. Very crudely, the GKL CA successively classifies "local" densities with the locality range increasing 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 both signals locally have $\rho = \rho_c$; the result is that the signal patterns can propagate, since the density of patterns with $\rho = \rho_c$ is not increased or decreased under the rule. In a simple sense, this is the CA's "strategy" for performing the computational task.

It has been claimed that the GKL CA performs the $\rho_c = 1/2$ task [18], but actually this is true only to an approximation. The GKL rule was invented not for the purpose of performing

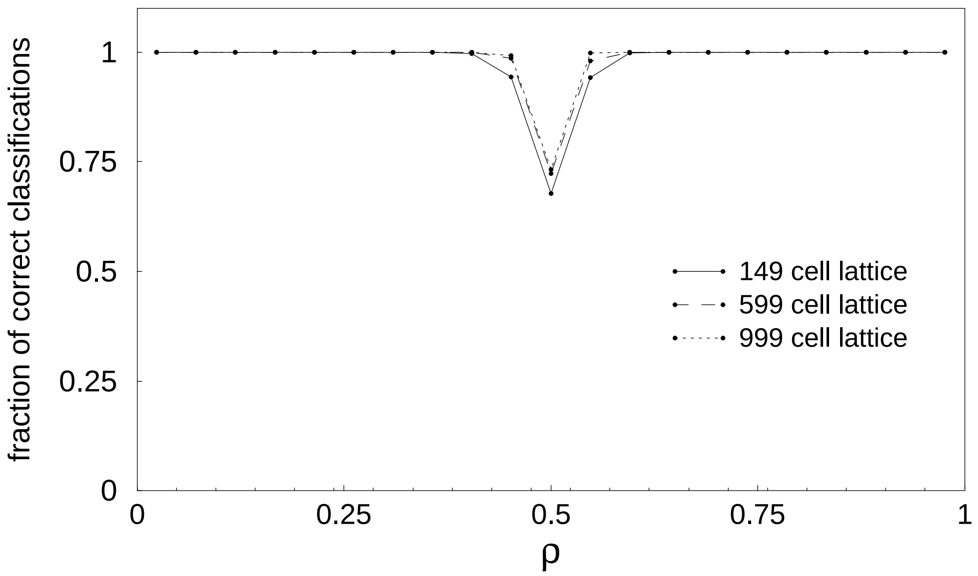


Figure 3: Experimental performance of the GKL rule as a function of $\rho(0)$ for the $\rho_c = 1/2$ task. Performance plots are given for three lattice sizes: $N = 149$ (the size of the lattice used in the GA runs), 599, and 999.

any particular computational task, but rather as part of studies of reliable computation and phase transitions in one spatial dimension. The goal in the former, for example, was to find a CA whose behavior is robust to small errors in the rule's update of the configuration. It has been proved that the GKL rule has only two attracting patterns, either all 1's or all 0's [6]. Attracting patterns here are those invariant patterns which, when perturbed a small amount, return to the same pattern. It turns out that the basins of attraction for the all-1 and all-0 patterns are not precisely the initial configurations with $\rho > 1/2$ or $\rho < 1/2$, respectively.[6] On finite lattices the GKL rule does classify most initial configurations according to this criterion, but on a significant number the "incorrect" attractor is reached. One set of experimental measures of the GKL CA's classification performance is displayed in Figure 3. To make this plot, we ran the GKL CA on 500 randomly generated initial configurations close to each of 21 densities $\rho \in [0.0, 1.0]$. The fraction of correct classifications was then plotted at each ρ. The rule was run either until a fixed point was reached or for a maximum number of time steps equal to $10 \times N$. This was done for CA with three different lattice sizes: $N \in \{149, 599, 999\}$.

Note that approximately 30% of the initial configurations with $\rho \approx \rho_c$ were misclassified. All the incorrect classifications are made for initial configurations with $\rho \approx \rho_c$. In fact, the worst performances occur at $\rho = \rho_c$. The error region narrows with increasing lattice size.

The GKL rule table has $\lambda = 1/2$, not $\lambda = \lambda_c$. Since it appears to perform a computational task of some complexity, at a minimum it is a deviation from the "edge of chaos" hypothesis for CA computation. The GKL rule's $\lambda = 1/2$ puts it right at the center of the "chaotic" region in Figure 2. This may seem puzzling since clearly the GKL rule does not produce chaotic behavior during either its transient or asymptotic epochs—far from it, in fact. However, the λ parameter was intended to correlate with "average" behavior of

[6]The terms "attractor" and "basin of attraction" are being used here in the sense of [6] and [12]. This differs substantially from the notion used in [34], for example. There "attractor" refers to any invariant or time-periodic pattern, and "basin of attraction" means that set of finite lattice configurations relaxing to it.

CA rules at a given λ value. Recall that γ in Figure 2 represents an *average* over a large number of randomly chosen CA rules and, while not shown in that plot, for most λ values the variance in γ is high. Thus, as was mentioned above, the behavior of any *particular* rule at its λ value might be very different from the *average* behavior at that value.

More to the point, though, we *expect* a λ value close to 1/2 for a rule that performs well on the $\rho_c = 1/2$ task. This is largely because the task is symmetric with respect to the exchange of 1's and 0's. Suppose, for example, a rule that carries out the $\rho_c = 1/2$ task has $\lambda < 1/2$. This implies that there are more neighborhoods in the rule table that map to output bit 0 than to output bit 1. This, in turn, means that there will be *some* initial configurations with $\rho > \rho_c$ on which the action of the rule will *decrease* the number of 1's. And this is the opposite of the desired action. However, if the rule acts to *decrease* the number of 1's on an initial configuration with $\rho > \rho_c$, it risks producing an intermediate configuration with $\rho < \rho_c$, which then would lead (under the original assumption that the rule carries out the task correctly) to a fixed point of all 0's, misclassifying the initial configuration. A similar argument holds in the other direction if the rule's λ value is greater than 1/2. This informal argument shows that a rule with $\lambda \neq 1/2$ will misclassify certain initial configurations. Generally, the further away the rule is from $\lambda = 1/2$, the more such initial configurations there will be. Such rules may perform fairly well, classifying most initial configurations correctly. However, we expect any rule that performs reasonably well on this task—in the sense of being close to the GKL CA's average performance shown in Figure 3—to have a λ value close to 1/2.

This analysis points to a problem with using this task as an evolutionary goal in order to study the relationship among evolution, computation, and λ. As was shown in Figure 2, for $r = 3, k = 2$ CA the λ_c values occur at roughly 0.23 and 0.83, and one hypothesis that was to be tested by the original experiment is that the GA will tend to select rules close to these λ_c values. But for the ρ-classification tasks, the range of λ values required for good performance is simply a function of the task and, specifically, of ρ_c. For example, the underlying 0-1 exchange symmetry of the $\rho_c = 1/2$ task implies that if a CA exists to do the task at an acceptable performance level, then it has $\lambda \approx 1/2$. Even though this does not directly invalidate the adaptation hypothesis or claims about λ's correlation with *average* behavior, it presents problems with using ρ-classification tasks as a way to gain evidence about a generic relation between λ and computational capability.

6.2 The Original Experiment

Packard used a GA to evolve CA rules to perform the $\rho_c = 1/2$ task. His GA started out with a randomly generated initial population of CA rules. Each rule was represented as a bit string containing the output bits of the rule table. That is, the bit at position 0 (i.e., the leftmost position) in the string is the state to which the neighborhood 0000000 is mapped, the bit at position 1 in the string is the state to which the neighborhood 0000001 is mapped, and so on. The initial population was randomly generated but it was constrained to be uniformly distributed across λ values between 0.0 and 1.0.

A given rule in the population was evaluated for ability to perform the classification task by choosing an initial configuration at random, running the CA on that initial configuration for some specified number of time steps, and at the final time step measuring the fraction of cells in the lattice that have the correct state. For initial configurations with $\rho > \rho_c$, the correct final state for each cell is 1, and for initial configurations with $\rho < \rho_c$, the correct final state for each cell is 0. For example, if the CA were run on an initial configuration with $\rho > \rho_c$ and at the final time step the lattice contained 90% 1's, the CA's score on that initial configuration would be 0.9.[7] The fitness of a rule was simply the rule's average score over a set of initial configurations. For each rule in the population, Packard generated a set of initial configurations that were uniformly distributed across ρ values from 0 to 1.

Packard's GA worked as follows. At each generation:

1. The fitness of each rule in the population is calculated.
2. The population is ranked by fitness.
3. Some fraction of the lowest fitness rules are removed.
4. The removed rules are replaced by new rules formed by crossover and mutation from the remaining rules.

Crossover between two strings involves randomly selecting a position in the strings and exchanging parts of the strings before and after that position. Mutation involves flipping one or more bits in a string, with some low probability.

A diversity-enforcement scheme was also used to prevent the population from converging too early and losing diversity [23]. If a rule is formed that is too close in Hamming distance (i.e., the number of matching bits) to existing rules in the population, its fitness is decreased.

The results from Packard's experiment are displayed in Figure 4. The two histograms display the observed frequency of rules in the GA population as a function of λ, with rules merged from a number of different runs. The top graph gives this data for the initial generation. As can be seen, the rules are uniformly distributed over λ values. The middle graph gives the same data for the final generation—in this case, after the GA has run for 100 generations. The rules now cluster around the two λ_c regions, as can be seen by comparison with the difference-pattern spreading rate plot, reprinted here at the bottom of the figure. Note that each individual run produced rules at one or the other peak in the middle graph, so when the runs were merged together, both peaks appear [23]. Packard interpreted these results as evidence for the hypothesis that, when an ability for complex computation is required, evolution tends to select rules near the transition to chaos. He argues, like Langton, that this result intuitively makes sense because "rules near the transition to chaos have the capability to selectively communicate information with complex structures in space-time, thus enabling computation." [24]

[7] A slight variation on this method was used in [24]. Instead of measuring the fraction of correct states in the final lattice, the GA measured the fraction of correct states over configurations from a small number n of final time steps [23]. This prevented the GA from evolving rules that were temporally periodic; viz. those with patterns that alternated between all 0's and all 1's. Such rules obtained higher than average fitness at early generations by often landing at the "correct" phase of the oscillation for a given initial configuration. That is, on the next time step the classification would have been incorrect. In our experiments we used a slightly different method to address this problem. This is explained in subsection 7.1.

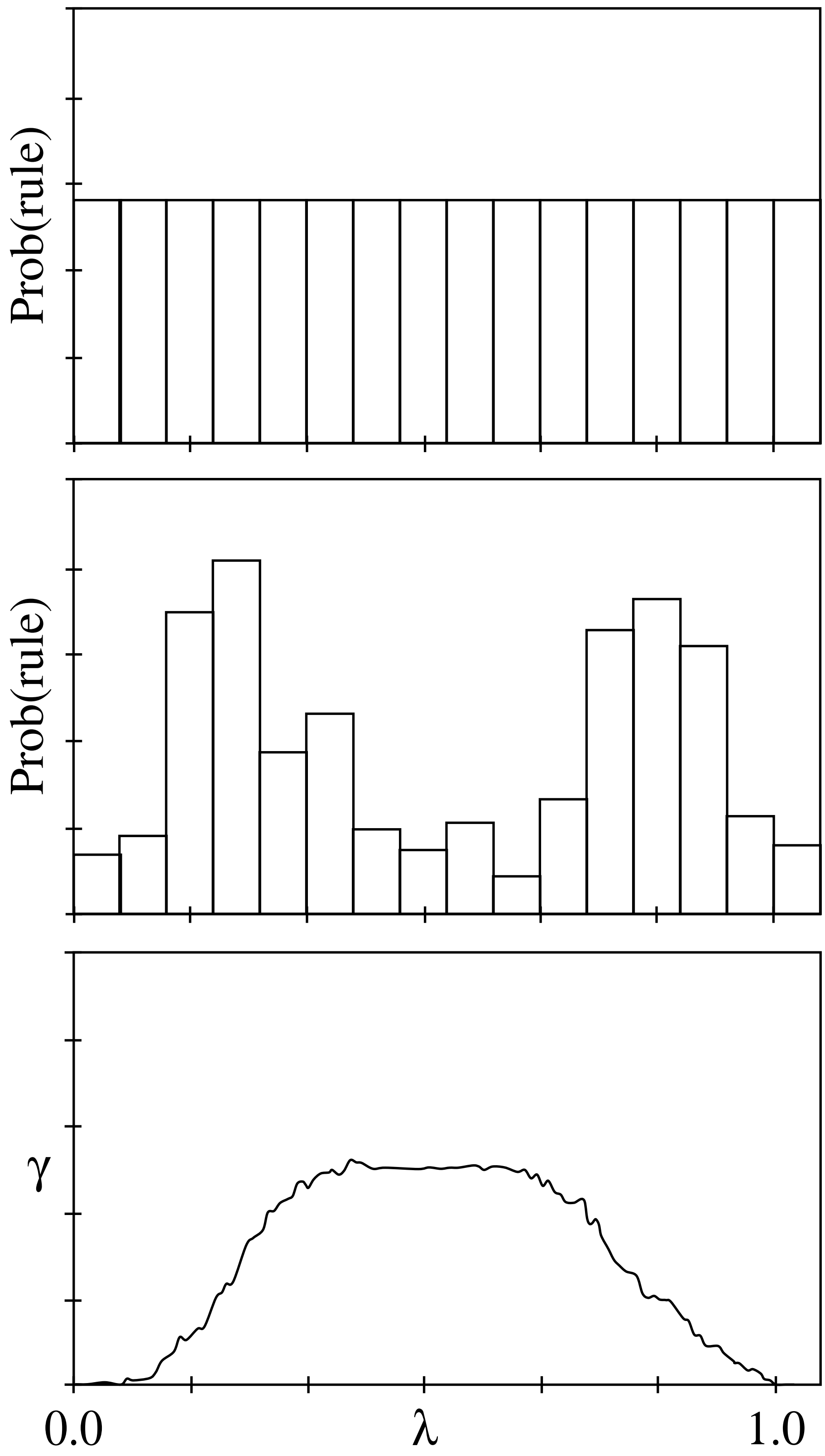


Figure 4: Results from the original experiment on GA evolution of CA for the $\rho_c = 1/2$ classification task. The top two figures are populations of CA at generations 0 and 100, respectively, versus λ. The bottom figure is Figure 2, reproduced here for reference. Adapted from [24], with permission of the author.

7. New Experiments

As the first step in a study of how well these general conclusions hold up, we carried out a set of experiments similar to that just described. We were unable to obtain some of the exact details of the original experiment's parameters, such as the exact population size for the GA, the mutation rate, and so on. As a result, we used what we felt were reasonable values for these various parameters. We carried out a number of parameter sensitivity tests which indicated that varying the parameters within small bounds did not change our qualitative results.

7.1 Details of Our Experiments

In our experiments, as in the original, the CA rules in the population all have $r = 3$ and $k = 2$. Thus the bit strings representing the rules are of length $2^{2r+1} = 128$ and the size of the search space is huge—the number of possible CA rules is 2^{128}. The tests for each CA rule are carried out on lattices of length $N = 149$ with periodic boundary conditions. The population size is 100, which was roughly the population size used in the original experiment [23]. The initial population is generated at random, but constrained to be uniformly distributed among different λ values. A rule's fitness is estimated by running the rule on 300 randomly generated initial configurations that are uniformly distributed over $\rho \in [0.0, 1.0]$. Exactly half the initial configurations have $\rho < \rho_c$ and exactly half have $\rho > \rho_c$.[8]

We allow each rule to run for a maximum number M of iterations, where a new M is selected for each rule from a Poisson distribution with mean 320. This is the measured maximum amount of time for the GKL CA to reach an invariant pattern over a large number of initial configurations on lattice size 149.[9] A rule's fitness is its average score—the fraction of cell states correct at the last iteration—over the 300 initial configurations. We term this fitness function *proportional fitness* to contrast with a second fitness function—*performance fitness*—which will be described below. A new set of 300 initial configurations is generated every generation. At each generation, all the rules in the population are tested on this set. Notice that this fitness function is stochastic—the fitness of a given rule may vary a small amount from generation to generation depending on the set of 300 initial configurations used in testing it.

[8] It was necessary to have this exact symmetry in the initial configurations at each generation to avoid early biases in the λ of selected rules. If, say, 49% of the initial configurations have $\rho < \rho_c$ and 51% of initial configurations have $\rho > \rho_c$, rules with λ close to 1 would obtain slightly higher fitness than rules with λ close to 0 since rules with λ close to 1 will map most initial configurations to all 1's. A rule with, say, $\lambda \approx 1$ would in this case classify 51% of the initial configurations correctly whereas a rule with $\lambda \approx 0$ would classify only 49% correctly. But such slight differences in fitness have a large effect in the initial generation, when all rules have fitness close to 0.5, since the GA selects the 50 *best* rules, even if they are only very slightly better than the 50 *worst* rules. This biases the representative rules in the early population. And this bias can persist well into the later generations.

[9] It may not be necessary to allow the maximum number of iterations M to vary. In some early tests with smaller sets of fixed initial configurations, though, we found the same problem Packard reported [23]: that if M is fixed, then period-2 rules evolve that alternate between all 0's and all 1's. These rules adapted to the small set of initial configurations and the fixed M by landing at the "correct" pattern for a given initial configuration at time step M, only to move to the opposite pattern and so wrong classification at time step $M + 1$. These rules did very poorly when tested on a different set of initial configurations—evidence for "over-fitting".

Our GA is similar to Packard's. In our GA, the fraction of new strings in the next generation—the "generation gap"—is 0.5. That is, once the population is ordered according to fitness, the top half of the population—the set of "elite" strings—is copied without modification into the next generation. For GA practitioners more familiar with nonoverlapping generations, this may sound like a small generation gap. However, since testing a rule on 300 "training cases" does not necessarily provide a very reliable gauge of what the fitness would be over a larger set of training cases, our selected gap is a good way of making a "first cut" and allowing rules that survive to be tested over more initial configurations. Since a new set of initial configurations is produced every generation, rules that are copied without modification are always retested on this new set. If a rule performs well and thus survives over a large number of generations, then it is likely to be a genuinely better rule than those that are not selected, since it has been tested with a large set of initial configurations. An alternative method would be to test every rule in every generation on a much larger set of initial configurations, but given the amount of compute time involved, that method seems unnecessarily wasteful. Much too much effort, for example, would go into testing very weak rules, which can safely be weeded out early using our method.

The remaining half of the population for each new generation is created by crossover and mutation from the previous generation's population.[10] Fifty pairs of parent rules are chosen at random with replacement from the entire previous population. For each pair, a single crossover point is selected at random, and two offspring are created by exchanging the subparts of each parent before and after the crossover point. The two offspring then undergo mutation. A mutation consists of flipping a randomly chosen bit in the string. The number of mutations for a given string is chosen from a Poisson distribution with a mean of 3.8 (this is equivalent to a per-bit mutation rate of 0.03). Again, to GA practitioners this may seem to be a high mutation rate, but one must take into account that at every generation, half the population is being copied without modification.

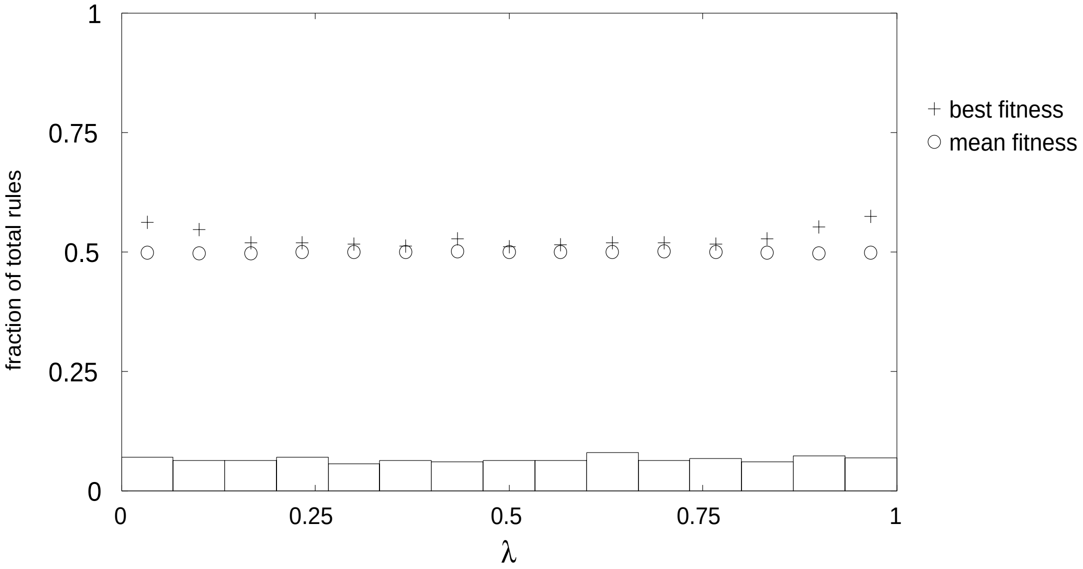
7.2 Results of Proportional-Fitness Experiment

We performed 30 different runs of the GA with the parameters described above, each with a different random-number seed. On each run the GA was iterated for 100 generations. We found that running the GA for longer than this, up to 300 generations, did not result in improved fitness. The results of this set of runs are displayed in Figure 5. Figure 5(a) is a histogram of the frequency of rules in the initial populations as a function of λ, merging together the rules from all 30 initial populations; thus the total number of rules represented in this histogram is 3000. The λ bins in this histogram are the same ones that were used by Packard, each of width 0.0667. Packard's highest bin contained only rules with $\lambda = 1$, that is, rules that consist of all 1's. We have merged this bin with the immediately lower bin.

As was said earlier, the initial population consists of randomly generated rules uniformly spread over the λ values between 0.0 and 1.0. Also plotted are the mean and best fitness values for each bin. These are all around 0.5, which is expected for a set of randomly

[10]This method of producing the non-elite strings differs from that in [24], where the non-elite strings were formed from crossover and mutation among the elite strings only rather than from the entire population. We observed no statistically significant differences in our tests using the latter mechanism other than a modest difference in time scale.

(a)



(b)

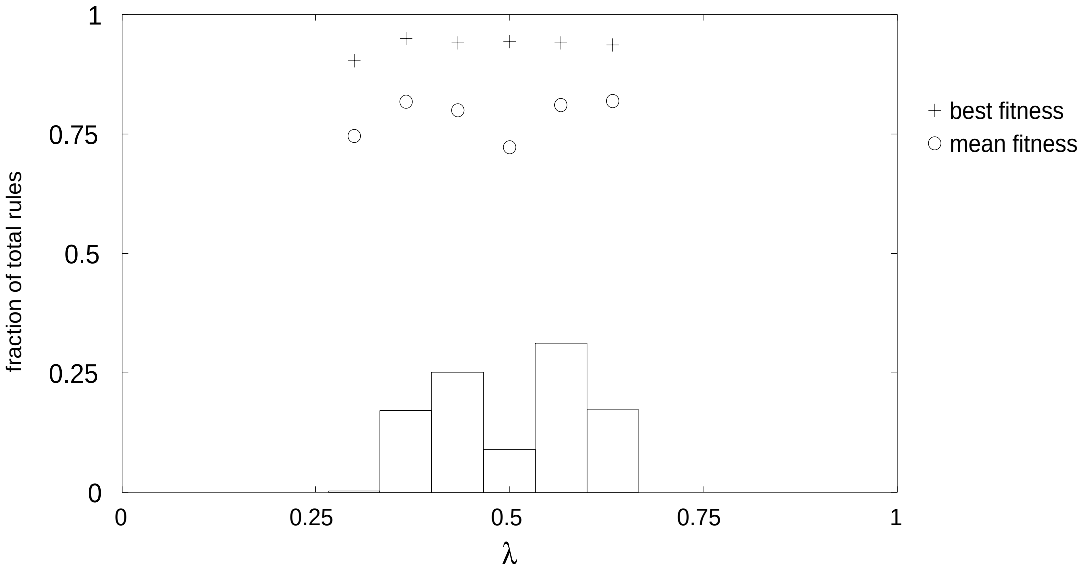


Figure 5: Results from our experiment with proportional fitness. The top histogram (a) plots as a function of λ the frequencies of rules merged from the initial generations of 30 runs. The bottom histogram (b) plots the frequencies of rules merged from the final generations (generation 100) of these 30 runs. Following [24] the x-axis is divided into 15 bins of length 0.0667 each. The rules with $\lambda = 1.0$ are included in the rightmost bin. In each histogram the best (cross) and mean (circle) fitnesses are plotted for each bin. (The y-axis interval for fitnesses is also [0,1]).

generated rules under this fitness function. The best fitnesses are slightly higher in the very low and very high λ bins. This is because rules with output bits that are almost all 0's (or 1's) correctly classify all low density (or all high density) initial configurations. In addition these CA obtain small partial credit on some high density (low density) initial configurations. Such rules thus have fitness sightly higher than 0.5.

Figure 5(b) shows the histogram for the final generation (100), merging together rules from the final generations of all 30 runs. Again the mean and best fitness values for each bin are plotted.

In the final generation the mean fitnesses in each bin are all around 0.8. The exceptions are the central bin with a mean fitness of 0.72 and the leftmost bin with a mean fitness of 0.75. The leftmost bin contains only five rules—each at $\lambda \approx 0.33$, right next to the the bin's upper λ limit. The standard deviations of mean fitness for each bin, not shown in the figure, are all approximately 0.15, except the leftmost bin, which has a standard deviation of 0.20. The best fitnesses for each bin are all between 0.93 and 0.95, except the leftmost bin which has a best fitness of 0.90. Under this fitness function the GKL rule has fitness ≈ 0.98; the GA never found a rule with fitness above 0.95.

As was mentioned above, the fitness function is stochastic: a given rule might be assigned a different fitness each time the fitness function is evaluated. The standard deviation under the present fitness scheme on a given rule is approximately 0.015. This indicates that the differences among the best fitnesses plotted in the histogram are not significant, except for that in the leftmost bin.

The lower mean fitness in the central bin is due to the fact that the rules in that bin largely come from non-elite rules generated by crossover and mutation in the final generation. This is a combinatorial effect: the density of CA rules as a function of λ is very highly peaked about $\lambda = 1/2$, as already noted. We will return to this "combinatorial drift" effect shortly. Many of the rules in the middle bin have not yet undergone selection and thus tend to have lower fitnesses than rules that have been selected in the elite. This effect disappears in Figure 6, which includes only the elite rules at generation 100 for the 30 runs. As can be seen, the difference in mean fitness disappears and the height of the central bin is decreased by half.

The results presented in Figure 5(b) are strikingly different from the results of the original experiment. In the final generation histogram in Figure 4, most of the rules clustered around either $\lambda \approx 0.24$ or $\lambda \approx 0.83$. In Figure 5(b), though, there are no rules in these λ_c regions. Rather, the rules cluster much closer—with a ratio of variances of 4 between the two distributions—to $\lambda \approx 0.5$. Recall this clustering is what we expect from the basic 0-1 exchange symmetry of the $\rho_c = 1/2$ task.

One rough similarity is the presence of two peaks centered around a dip at $\lambda \approx 0.5$—a phenomenon which we will explain shortly and which is a key to understanding how the GA is working. But there are significant differences, even within this similarity. In the original experiments the peaks are in bins centered about $\lambda \approx 0.23$ and $\lambda \approx 0.83$. In Figure 5(b), though, the peaks are very close to $\lambda = 1/2$, being centered in the neighboring bins—those with $\lambda \approx 0.43$ and $\lambda \approx 0.57$. Thus, the ratio of original to current peak spread is roughly a factor of 4. Additionally, in the final-generation histogram of Figure 4 the two highest bin populations are roughly five times as high as the central bin, whereas in Figure 5(b) the two

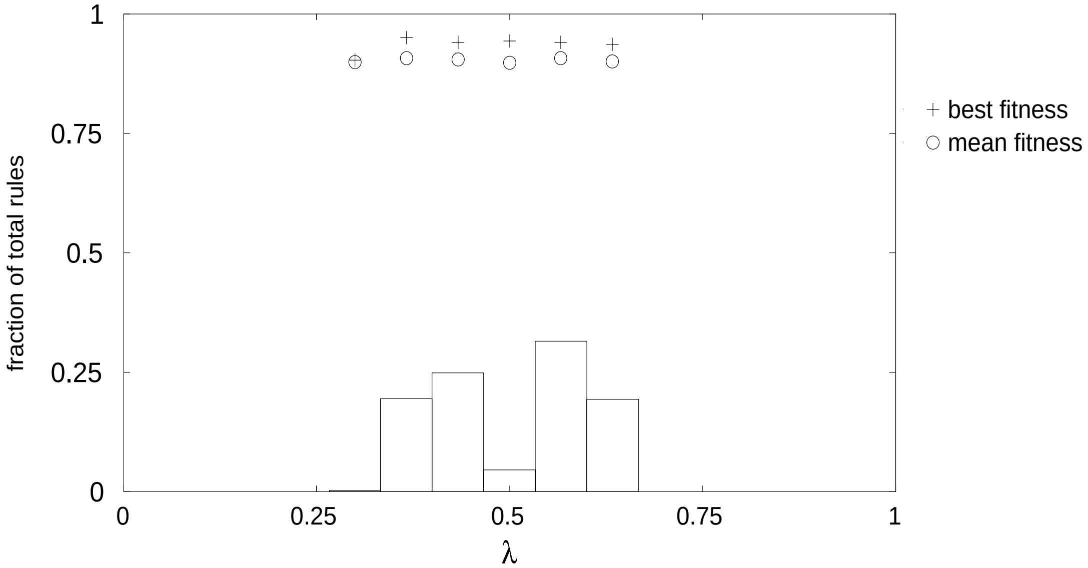


Figure 6: Histogram including only the elite rules from the final generations of the 30 runs (cf. Figure 5(b)) with the proportional-fitness function.

highest bins are roughly three times as high as the central bin. Finally, the final-generation histogram in Figure 4 shows the presence of rules in every bin, but in Figure 5(b), there are rules only in six of the central bins.

Similar to the original experiment, we found that on any given run the population was clustered about one or the other peak but not both. Thus, in the histograms that merge all runs, two peaks appear. This is illustrated in Figure 7, which displays histograms from the final generation of two individual runs. In one of these runs the population clustered to the left of the central bin, in the other run it clustered to the right of the center. The fact that different runs result in different clustering locations is why we performed many runs and merged the results rather than performing a single run with a much larger population. The latter method might have yielded only one peak. Said a different way, independent of the population size a given run will be driven by and the population organized around the fit individuals that appear earliest. Thus, examining an ensemble of individual runs reveals more details of the evolutionary dynamics.

The asymmetry in the heights of the two peaks in Figure 5(b) results from a small statistical asymmetry in the results of the 30 runs. There were 14 out of 30 runs in which the rules clustered at the lower λ bin and 16 out of 30 runs in which the rules clustered at the higher λ bin. This difference is not significant, but explains the small asymmetry in the peaks' heights.

We extended 16 of the 30 runs to 300 generations, and found that not only do the fitnesses not increase further, but the basic shape of the histogram does not change significantly.

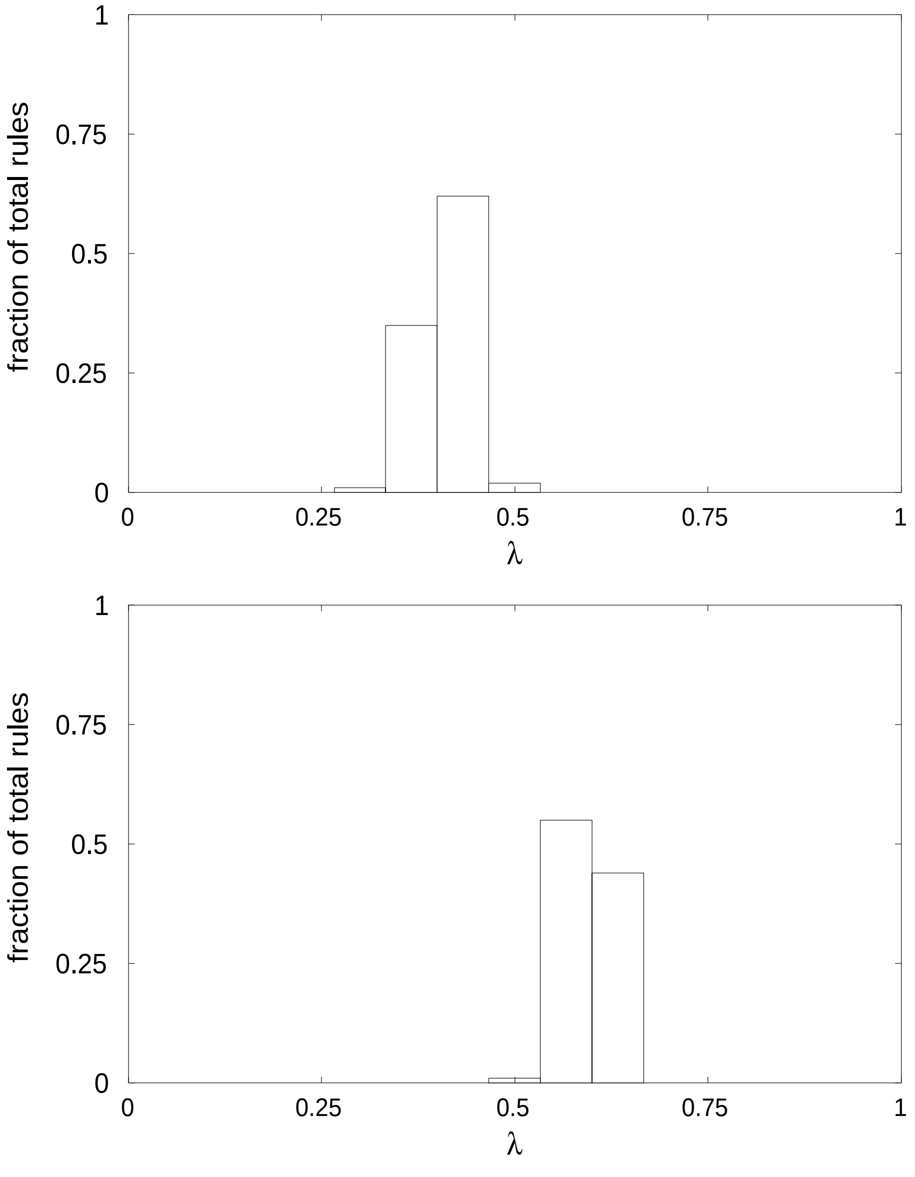


Figure 7: Histograms from the final generations of two individual runs of the GA employing proportional fitness. Each run had a population of 100 rules. The final distribution of rules in each of the 30 runs we performed resembled one or the other of these two histograms.

7.3 Effects of Drift

The results of our experiments suggest that, for the $\rho_c = 1/2$ task, an evolutionary process modeled by a genetic algorithm tends to select rules with $\lambda \approx 1/2$. This is what we expect from the theoretical discussion given above concerning this task and its symmetries. We will delay until the next section a discussion of the curious feature near $\lambda = 1/2$, viz. the dip surrounded by two peaks. Instead, here we focus on the larger-scale clustering in that λ region.

To understand this clustering we need to understand the degree to which the selection of rules close to $\lambda = 1/2$ is due to an intrinsic selection pressure and the degree to which it is due to "drift". By "drift" we refer to the force that derives from the combinatorial aspects of CA space as explored by random selection ("genetic drift") along with the effects of crossover and mutation. The intrinsic effects of random selection with crossover and mutation are to move the population, irrespective of any selection pressure, to $\lambda = 1/2$. This is illustrated by the histogram mosaic in Figure 8. These histograms show the frequencies of the rules in the population as a function of λ every 5 generations, with rules merged from 30 runs on which selection according to fitness was turned off. That is, on these runs, the fitness of the rules in the population was never calculated, and at each generation the selection of the elite group of strings was performed at random. Everything else about the runs remains the same as before. Since there is no fitness-based selection, drift is the only force at work here. As can be seen, under the effects of random selection, crossover, and mutation, by generation 10 the population has largely drifted to the region of $\lambda = 1/2$ and this clustering becomes increasingly pronounced as the run continues.

This drift to $\lambda = 1/2$ is related to the combinatorics of the space of bit strings. For binary CA rules with neighborhood size n ($= 2r + 1$), the space consists of all 2^{2^n} binary strings of length 2^n. Denoting the subspace of CA with a fixed λ and n as $\mathrm{CA}(\lambda, n)$, we see that the size of the subspace is binomially distributed with respect to λ:

$$|\mathrm{CA}(\lambda, n)| = \binom{2^n}{\lambda 2^n}.$$

The distribution is symmetric in λ and tightly peaked about $\lambda = 1/2$ with variance $\propto 2^{-n}$. Thus, the vast majority of rules is found at $\lambda = 1/2$. The steepness of the binomial distribution near its maximum gives an indication of the magnitude of the drift "force". Note that the last histogram in Figure 8 gives the GA's rough approximation of this distribution.

Drift is thus a powerful force moving the population to cluster around $\lambda = 1/2$. For comparison, Figure 9 gives the rule-frequency-versus-λ histograms for the 30 runs of our proportional-fitness experiment every five generations. The last histogram in this figure is the same one that was displayed in Figure 5(b). (Figure 9 gives the merged data from the entire population of each run every five generations. A similar mosaic plotting only the elite strings at each generation looks qualitatively similar.)

Figure 9 looks roughly similar to Figure 8 up to generation 35. The main difference in generations 0–30 is that Figure 9 indicates a more rapid peaking about $\lambda = 1/2$. The increased speed of movement to the center over that seen in Figure 8 is presumably due to the additional evolutionary pressure of proportional fitness. At generation 35, something new appears. The peak in the center has begun to shrink significantly and the two surrounding