A Genetic Algorithm Discovers Particle-Based Computation in Cellular Automata

Rajarshi Das[1], Melanie Mitchell[1], and James P. Crutchfield[2]

[1] Santa Fe Institute, 1660 Old Pecos Trail, Suite A, Santa Fe, New Mexico, U.S.A. 87501.

[2] Physics Department, University of California, Berkeley, CA, U.S.A. 94720.

In Y. Davidor, H.-P. Schwefel, and R. Männer (editors), *Parallel Problem Solving from Nature—PPSN III*. Berlin: Springer-Verlag.

Abstract. How does evolution produce sophisticated emergent computation in systems composed of simple components limited to local interactions? To model such a process, we used a genetic algorithm (GA) to evolve cellular automata to perform a computational task requiring globally-coordinated information processing. On most runs a class of relatively unsophisticated strategies was evolved, but on a subset of runs a number of quite sophisticated strategies was discovered. We analyze the emergent logic underlying these strategies in terms of information processing performed by "particles" in space-time, and we describe in detail the generational progression of the GA evolution of these strategies. Our analysis is a preliminary step in understanding the general mechanisms by which sophisticated emergent computational capabilities can be automatically produced in decentralized multiprocessor systems.

1. Introduction

Natural evolution has created many systems in which the actions of simple, locally-interacting components give rise to coordinated global information processing. Insect colonies, economic systems, the immune system, and the brain have all been cited as examples of systems in which "emergent computation" occurs (e.g., see [5, 8]). In the following, "emergent computation" refers to the appearance in a system's temporal behavior of information-processing capabilities that are neither explicitly represented in the system's elementary components or their couplings nor in the system's initial and boundary conditions. Our interest is in phenomena in which many locally-interacting processors, unguided by a central control, result in globally-coordinated information processing that is more powerful than can be done by individual components or linear combinations of components. More precisely, "emergent computation" signifies that the global information processing can be interpreted as implementing (or approximating) a computation [1].

While observations of the behavior of such a decentralized, multicomponent system might suggest that a computation is taking place, understanding the emergent logic by which the computation is performed is typically very difficult. It is also not well understood how evolution creates the capacity for emergent computation in such systems. In this paper we report work addressing both these questions in a simplified model in which emergent computational capabilities in cellular automata are evolved under a genetic algorithm (GA). We apply a framework, called "computational mechanics", for analyzing the emergent logic embedded in the spatiotemporal behavior of spatially-extended systems such as cellular automata [3]. The results demonstrate how globally-coordinated information processing is mediated by "particles" and "particle interactions" in space-time. We also analyze in detail the evolutionary history over which a GA evolved such emergent computation in cellular automata. Though

our model is simplified, the results are relevant to understanding the evolution of emergent computation in more complicated systems.

2. Cellular Automata

One of the simplest systems in which emergent computation can be studied is a one-dimensional binary-state cellular automaton (CA)—a one-dimensional lattice of N two-state machines ("cells"), each of which changes its state as a function only of the current states in a local neighborhood. The lattice starts out with an initial configuration of cell states (0s and 1s) and this configuration changes in discrete time steps according to the CA "rule". We use the term "state" to refer to a local state s_i—the value of the single cell at site i. The term "configuration" will refer to the pattern of local states over the entire lattice.

A CA rule ϕ can be expressed as a look-up table ("rule table") that lists, for each local neighborhood, the update state for the neighborhood's central cell. In a one-dimensional CA, a neighborhood consists of a cell and its r ("radius") neighbors on either side. A sample rule (the "majority" rule) for a one-dimensional binary CA with $r = 1$ is the following. Each possible neighborhood η is given along with the "output bit" $s = \phi(\eta)$ to which the central cell is updated. (The neighborhoods are listed in lexicographic order.)

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

At time step t, the look-up table is applied to each neighborhood in the current lattice configuration, respecting the choice of boundary conditions, to produce the configuration at $t + 1$.

Cellular automata have been studied extensively as mathematical objects, as models of natural systems, and as architectures for fast, reliable parallel computation (for an overview of CA theory and applications, see, e.g., [12, 13]). However, the difficulty of understanding the emergent behavior of CAs or of designing CAs to have desired behavior has up to now severely limited their applications in science and engineering, and for general computation.

3. Details of Experiments

We used a form of the GA to evolve one-dimensional, binary-state CAs to perform a density classification task: the $\rho_c = 1/2$ task [9]. A successful CA for this task will decide whether or not the initial configuration (IC) contains more than half 1s. Let ρ denote the density of 1s in a configuration and let ρ_0 denote the density of 1s in the initial configuration. If $\rho_0 < \rho_c$ ($\rho_0 > \rho_c$), then within M time steps the CA should relax to the fixed-point configuration of all 0s (1s). M is a parameter of the task that depends on the lattice size N.

The CAs in our experiments had $r = 3$, with spatially periodic boundary conditions: $s_i = s_{i+N}$. The $\rho_c = 1/2$ 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 mediated by the cell neighborhoods. The minimum amount of memory required for the $\rho_c = 1/2$ task 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. It has been argued that no finite radius CA can perform this

task perfectly across all lattice sizes (C. Moore, personal communication), but even to perform this task well for a fixed lattice size requires more powerful computation than can be performed by a single cell or any linear combination of cells, such as the majority rule. Since the 1s can be distributed throughout the CA lattice, the CA must transfer information over large space-time distances ($\approx N$).

The GA begins with a population of P randomly generated 128-bit CA rules, each encoded as a string of the rule-table output bits in lexicographic order of neighborhood patterns. (There are 2^{128} possible rules—far too many 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 $\rho \in [0.0, 1.0]$, exactly half with $\rho < \rho_c$ and half with $\rho > \rho_c$; (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., N 0s for $\rho_0 < \rho_c$ and N 1s for $\rho_0 > \rho_c$. The initial density ρ_0 is never exactly 1/2, since N is chosen to be odd. Rule ϕ's fitness $F_I(\phi)$ is the fraction of the I ICs on which ϕ produces the correct final pattern. No partial credit is given for partially correct final configurations.

It should be pointed out that sampling ICs with uniform distribution over $\rho \in [0.0, 1.0]$ is highly skewed with respect to an unbiased distribution of ICs, which is binomially distributed over $\rho \in [0.0, 1.0]$ and very strongly peaked at $\rho = 1/2$. However, preliminary experiments indicated a need for such a biased distribution in order for the GA to make progress in early generations. This biased distribution turns out to impede the GA in later generations because, as increasingly fitter rules are evolved, the IC sample becomes less and less challenging for the GA [9].

The GA works as follows. (i) A new set of I ICs is generated. (ii) $F_I(\phi)$ is calculated for each rule ϕ in the population. (iii) A number E of the highest fitness ("elite") rules is copied without modification to the next generation. (iv) The remaining $P-E$ rules for the next generation are formed by single-point crossovers between pairs of elite rules randomly chosen with replacement. The offspring from each crossover are each mutated m times, where mutation consists of flipping a randomly chosen bit in a string. This defines one generation of the GA; it is repeated G times for one run of the GA. This method is similar to that used by Packard to evolve CAs for the $\rho_c = 1/2$ task [11]. (For a discussion of Packard's experiment, see [10].) Note that F_I is a random variable, 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 we used were the following. For each CA in the population, $N = 149$ and $I = 100$. Each time a CA was simulated, M was chosen from a Poisson distribution with mean 320 (slightly greater than $2N$). Varying M prevents overfitting of rules to a particular M; see [10]. Allowing M to be larger—up to ten times the lattice size—did not change the qualitative results of the experiments [9]. The chromosomes in the initial population were not chosen with uniform probability at each bit as is common practice, but rather were uniformly distributed over the fraction of 1s in the string. (This restriction for the initial population was made for reasons related to previous research questions; see [10]. A smaller set of subsequent experiments with unbiased randomly generated initial populations indicated that this restriction is not likely to significantly influence the results of the experiments.) We set $P = 100$; $E = 20$; $m = 2$; and $G = 50$ (in some runs G was set to 100; no significant

CA	Rule Table	$N = 149$	$N = 599$	$N = 999$
Majority	ϕ_{maj}	0.000	0.000	0.000
Expand 1-Blocks	ϕ_{1a}	0.652	0.515	0.503
Particle-Based	ϕ_{1b}	0.697	0.580	0.522
Particle-Based	ϕ_{1c}	0.742	0.718	0.701
Particle-Based	ϕ_{1d}	0.769	0.725	0.714
GKL	ϕ_{GKL}	0.816	0.766	0.757

Table 1: Measured values of $\mathcal{P}^N_{10^4}$ at various values of N for six different $r = 3$ rules: the majority rule, the four rules discovered by the GA in different runs (ϕ_{1a}–ϕ_{1d}), and the GKL rule. The subscripts for the discovered rule tables indicate the pair of space-time diagrams illustrating their behavior in Figure 1. The standard deviation of $\mathcal{P}^{149}_{10^4}$, when calculated 100 times for the same rule, is approximately 0.004. The standard deviations for $\mathcal{P}^N_{10^4}$ for larger N are higher. (This table is similar to that given in [4], where the complete look-up tables for these rules are also given.)

difference in the final results was observed). For a more detailed justification of these parameter settings and the results of parameter-modification experiments, see [9, 10].

4. Results of Experiments

4.1 Previous Results

We performed 300 runs of the GA with the parameters given above; each run had a different random-number seed. On most runs, the GA proceeded through roughly the same sequence of four "epochs" of innovation, each of which was marked by the discovery of a significantly improved new strategy for performing the $\rho_c = 1/2$ task. As reported in [9, 10], on most runs the GA evolved one of two strategies: (1) Relax to the fixed point of all 0s unless there is a sufficiently large ($\sim 2r + 1$) block of adjacent (or almost adjacent) 1s in the IC. If so, expand that block. (2) Relax to the 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.

A rule implementing strategy (1)—here called ϕ_{1a}—is illustrated in Figure 1(a). The figure gives two "space-time diagrams"—plots of lattice configurations over a range of time steps, with 1s given as black cells, 0s given as white cells, and time increasing down the page. Strategies (1) and (2) rely on the appearance or absence of blocks of 1s or 0s in the IC to be good predictors of ρ_0. For example, high-ρ ICs are more likely to have blocks of adjacent 1s than low-ρ ICs (cf. the bottom diagram in Figure 1(a)). The size of blocks that are expanded was tuned by the GA to be a good predictor of high or low density for $N = 149$ given the distribution of ICs on which the rules were tested.

The block-expanding rules evolved by the GA do not count as sophisticated examples of computation in CAs: all the computation is done locally in identifying and then expanding a "sufficiently large" block. Under F_{100} these strategies obtained fitnesses between 0.9 and 1.0 for different sets of ICs. A more indicative performance measure is "unbiased performance", $\mathcal{P}^N_I(\phi)$, defined as the fraction of I ICs chosen from an unbiased distribution over ρ on which rule ϕ produces the correct final pattern after $2.15N$ time steps. (With an unbiased distribution, most ICs chosen have $\rho \approx 0.5$. These are the hardest cases for any rule to classify.) After each run of the GA we measured $\mathcal{P}^{149}_{10^4}$ for the elite rules in the final generation. The highest measured $\mathcal{P}^{149}_{10^4}(\phi)$ for the block-expanding rules was approximately 0.685. The performance of these rules decreased dramatically for larger N since the size of block to expand was

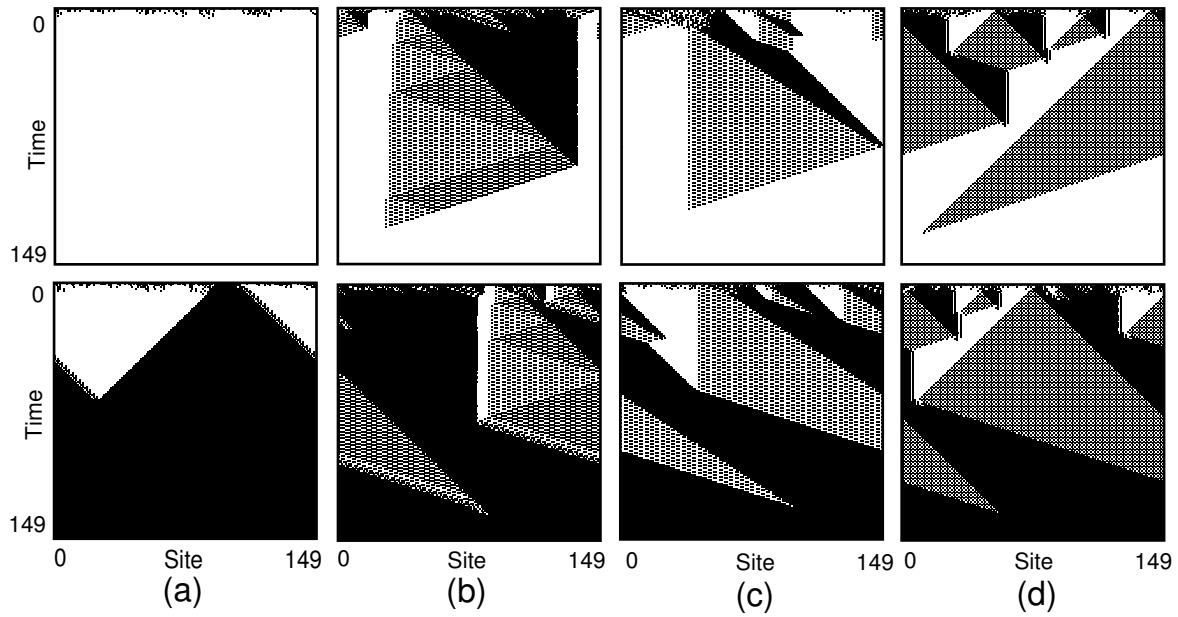

Figure 1: Space-time diagrams for four different rules discovered by the GA. The top diagrams have $\rho_0 < 1/2$; the bottom diagrams have $\rho_0 > 1/2$. Fitness increases from (a) to (d). (a) ϕ_{1a}. (b) ϕ_{1b}. (c) ϕ_{1c}. (d) ϕ_{1d}. Note that the "grey" pattern in (d) consists of alternating 1s and 0s.

tuned by the GA for $N = 149$. The second row of Table 1 gives $\mathcal{P}^N_{10^4}(\phi_{1a})$ for three values of N. It is interesting to note that one naive solution to the $\rho_c = 1/2$ task—the $r = 3$ "majority" rule, in which the new state of each cell is decided by a majority vote among the $2r + 1$ cells in the neighborhood—has $\mathcal{P}^{149}_{10^4} = 0$ for all three values of N (first row of Table 1).

Mitchell, Crutchfield, and Hraber [9] described in detail the typical progression of the GA through the four epochs of innovation and the evolutionary mechanisms by which each new strategy was discovered on the way to a block-expanding CA. They also discussed a number of impediments that tended to keep the GA from discovering fitter, more general strategies; primary among them was the GA's breaking of the $\rho_c = 1/2$ task's symmetries in early generations for short-term gain by specializing exclusively on 1-block or 0-block expansion. On most runs this symmetry breaking was an important factor in preventing the GA from progressing to more sophisticated rules.

4.2 The New Strategies Discovered by the GA

Despite the impediments discussed in [9] and the unsophisticated rules evolved on most runs, on seven different runs the GA discovered rules with significantly higher performance and more sophisticated computational properties. Three such rules (each from a different run) are illustrated in Figures 1(b)–(d). For each of these rules—ϕ_{1b}, ϕ_{1c}, and ϕ_{1d}—F_{100} was between 0.9 and 1.0, depending on the set of 100 ICs. Some $\mathcal{P}^N_{10^4}$ values for these three rules are given in Table 1. As can be seen, $\mathcal{P}^{149}_{10^4}$ is significantly higher for these rules than for the typical block-expanding rule ϕ_{1a}. In addition, the performances of ϕ_{1c} and ϕ_{1d} remain relatively constant as N is increased, indicating a marked improvement in generalization.

Why does ϕ_{1d}, for example, perform relatively well on the $\rho_c = 1/2$ task? In Figure 1(d) 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 about the density in local regions takes place. This local information interacts with other local information to produce the desired final state. Roughly ϕ_{1d} 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 black-to-white 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.

The discovery of these rules was first reported in [4] and methods for understanding their emergent logic were sketched there. In the following we first briefly review the computational mechanics approach to CAs. Using these methods, we then analyze the strategy of ϕ_{1d}—the rule with highest $\mathcal{P}^N_{10^4}$ found in the 300 runs—and describe the generational progression by which it was evolved under the GA.

4.3 Computational Mechanics of Cellular Automata

Like many natural processes that are spatially-extended, cellular-automata configurations often organize over time into spatial regions that are dynamically homogeneous. Sometimes in space-time diagrams these regions are obvious to the eye as "domains": regions in which the same "pattern" appears. In order to understand this phenomenon, and also to deal with cases in which human observation is inadequate or undesired, the notion of "domain" was formalized in [7] by adapting computation theory to CA dynamics. There, a domain's "pattern" is described using the minimal deterministic finite automaton (DFA) that accepts all and only those configurations that appear in the domain. Such domains are called "regular" since their configurations are members of the regular language recognized by the DFA. More precisely, a regular domain Λ is a set of configurations that is temporally invariant ($\Lambda = \phi(\Lambda)$) and whose DFA has a single recurrent set of states that is strongly connected.

Regular domains play a key role in organizing both the dynamical behavior and the information-processing properties of CA. Once a CA's regular domains have been detected, for example, nonlinear filters can be constructed to filter them out, leaving just the deviations from those regularities. The resulting filtered space-time diagram reveals the propagation of domain "walls". If these walls remain spatially-localized over time, then they are called "particles" [3]. Particles are one of the main mechanisms for carrying information over long space-time distances. This information might indicate, for example, the result of some local processing which has occurred at an early time. In this way particles can serve as signals. Logical operations on the information they contain are performed when the particles interact. The collection of domains, domain walls, particles, and particle interactions for a CA represents the basic information-processing elements embedded in the CA's behavior—the CA's "intrinsic" computation.

4.4 The GA Discovery of Particle-Based Computation

In this section we apply the computational mechanics framework sketched above to describe in detail the route by which the GA discovered ϕ_{1d}.

Figure 2(a) plots best fitness (F_{100}) in the population versus generation for the first 50 generations of the run in which ϕ_{1d} was discovered. It can be seen that, before the GA discovers high fitness rules, the fitness of the best CA rule increases

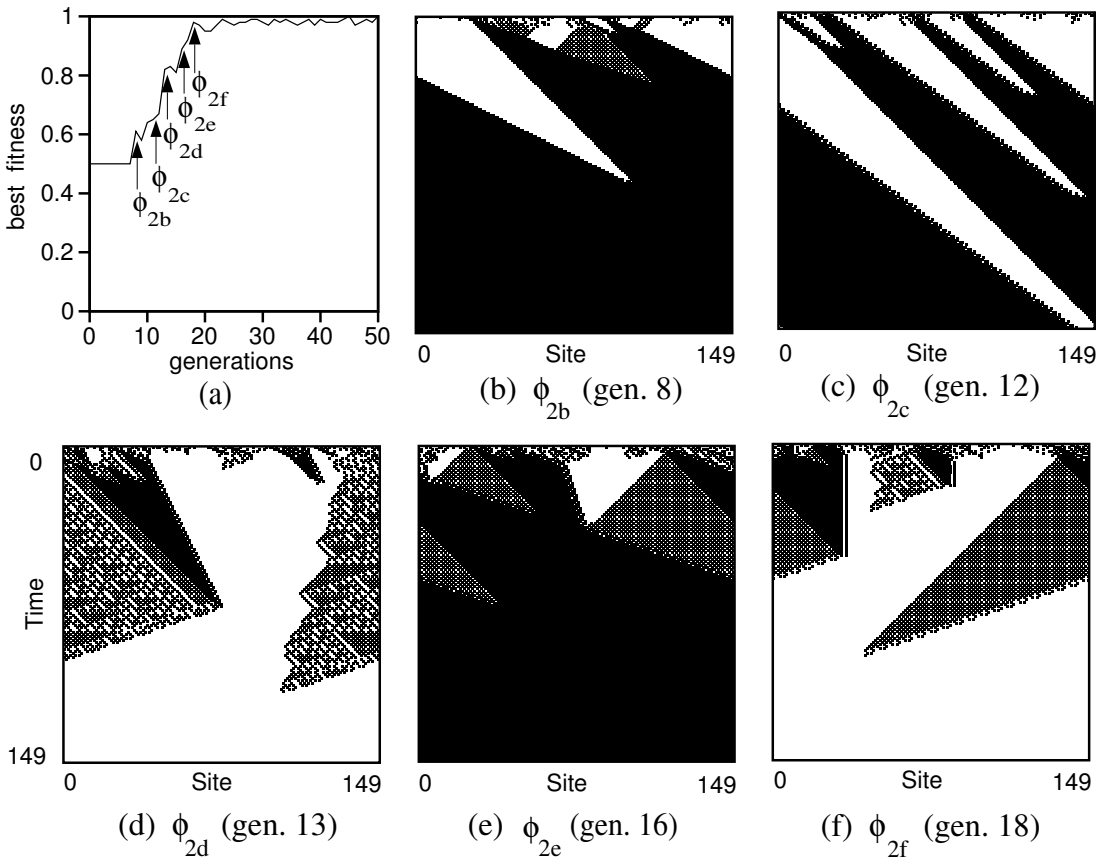

Figure 2: (a) Plot of the fitness of the most fit rule in the population versus generation in the run producing ϕ_{1d}. The arrows in the plot indicate the generations in which the GA discovered each new significantly improved strategy. (b)–(f) Space-time diagrams illustrating the behavior of the best rule at each of the five generations marked in (a).

in rapid jumps after periods of relative stasis. Qualitatively, the rise in performance can be divided into several "epochs", each beginning with the discovery of a new, significantly improved strategy for performing the $\rho_c = 1/2$ task. In Figure 2(a), the initial generations of these epochs are labeled with the name of the best rule found at that generation.

For the first seven generations, the highest F_{100} value equal to 0.5 is achieved by a number of rules that map almost all neighborhoods to the same symbol. Rules in which almost all neighborhoods map to 1 (0) quickly settle to the all 1s (all 0s) configuration irrespective of ρ_0. Such rules are able to correctly classify exactly half of the ICs and thus have a fitness of 0.5.

In generation 8, a new rule (ϕ_{2b}) is discovered, resulting in significantly better performance ($F_{100} \approx 0.61$). Its strategy is illustrated by the space-time diagram in Figure 2(b). ϕ_{2b} was created by a crossover between rule that maps most neighborhoods to 0 and a rule that maps most neighborhoods to 1. A closer inspection of the behavior of this rule reveals that it always relaxes to the all 1s configuration except when ρ_0 is close to zero, in which case it relaxes to the all 0s configuration, yielding $F_{100} > 0.5$. An analysis of particle interactions explains ϕ_{2b}'s behavior. After a very short transient phase, ϕ_{2b} essentially creates three regular domains: 1^* (denoted B),

0^* (denoted W), and $(10)^* \bigcup (01)^*$ (denoted #). This behavior can be understood readily from ϕ_{2b}'s rule table: out of the 28 neighborhood patterns with five or more 1s (0s), 27 (22) result in an output bit of 1 (0). Therefore in any IC, small "islands" of length 1, 2 or 3 containing only 1s or 0s in are quickly eliminated, resulting in locally uniform domains. To maintain a # domain, the neighborhood patterns (1010101) and (0101010) must produce a 1 and 0 respectively, as is done in ϕ_{2b}.

After the B, W, and # domains are created, the subsequent behavior of ϕ_{2b} is primarily determined by the interaction of the particles representing the domain walls. ϕ_{2b} has three domains and six particles, one for each domain wall type: $P_{BW}, P_{WB}, P_{B\#}, P_{\#B}, P_{W\#}$, and $P_{\#W}$. The velocities of these particles (i.e., the slope of the domain wall corresponding to the particle) are 2.0, 1.0, 3.0, 1.0, -1.0, and -1.0 respectively. There are two annihilative interactions: $P_{BW} + P_{WB} \rightarrow \emptyset_B$ and $P_{B\#} + P_{\#B} \rightarrow \emptyset_B$ (where $\emptyset_B$ indicates a B domain with no particles). Two interactions are reactive: $P_{BW} + P_{W\#} \rightarrow P_{B\#}$ and $P_{B\#} + P_{\#W} \rightarrow P_{BW}$. All these interactions can be seen in Figure 2(b). Several particle interactions never occur as a result of the direction and magnitude of the particle velocities. For example, the particles $P_{\#W}$ and $P_{W\#}$ have the same velocity and never interact. Similarly, although $P_{\#B}$ moves in the same direction as P_{BW}, the former has only half the speed of the latter, and the two particles never have the chance to interact. For similar reasons, the # domain and its four associated particles $P_{B\#}, P_{\#B}, P_{W\#}$, and $P_{\#W}$ play no appreciable role in the determination of the CA's final configuration. This is easily verified by complementing the output bit of one of the two neighborhood patterns necessary to maintain the # domain and observing that very little change occurs in the rule's fitness.

Thus, for this epoch we can focus exclusively on the boundaries between the B and the W regions. Because P_{WB}'s velocity is less than that of P_{BW}, P_{BW} soon catches up with P_{WB}. The interaction of these two walls results in a B domain, eliminating the W domain. Therefore when an island of W is surrounded by B domains, the size of the former shrinks until it vanishes from the lattice. Conversely, when an island of B is surrounded by W domains, the B domain grows until it occupies the whole lattice. However, an island of B must occupy at least five adjacent cells in order to expand. Thus if any initial configuration contains a block of five 1s *or* results in the creation of such a block when the rule is applied over subsequent time steps, then the CA inevitably relaxes to the all 1s configuration. Otherwise it relaxes to the all 0s configuration.

The explanation of ϕ_{2b}'s behavior in terms of particles and particle interactions may seem a complicated way to describe simple behavior. But the usefulness of the computational mechanics framework for explicating computation will become clearer as the space-time behavior becomes more complex.

The next four generations produce no new epochs (i.e., no significantly new strategies) but a modest improvement in the best fitness. During this period three changes in behavior were observed, all of which appear in the generation 12 rule ϕ_{2c} (see Figure 2(c)). First, for an island of B to expand and take over the whole lattice, it now must contain at least seven cells. Since a seven-cell island of B is less likely than a five-cell island in low-ρ ICs, more low-ρ ICs are correctly classified. Second, the velocity of $P_{\#W}$ is modified from -1.0 to -3.0, allowing the annihilative reaction: $P_{W\#} + P_{\#W} \rightarrow \emptyset_W$. Thus, unlike in ϕ_{2b}, an island of # domain when surrounded by the W domain cannot persist indefinitely. Third, the velocity of P_{BW} is decreased to 1.5. Since the velocity of P_{WB} remains constant at 1.0, it now takes longer to

eliminate W domains. (This last modification do not result in a significant change in fitness.) Unlike the innovation that resulted in ϕ_{2b}, where crossover played the major role, we find that these modifications (and the ones that follow) are primarily due to the mutation operator.

In generation 13, a new epoch begins with the discovery of ϕ_{2d}, with a sharp jump in F_{100} to 0.82 corresponding to a significant innovation. ϕ_{2d}'s behavior is illustrated in Figure 2(d). Here we use # to refer to an interesting variation of the checkerboard ($(10)^* \bigcup (01)^*$) domain. While the velocity of P_{WB} is held constant at 1.0, P_{BW} now moves with velocity 0.5 in the same direction. Since P_{BW} cannot catch up with P_{WB}, an island of W can now expand when surrounded by B, with the condition that W has to be at least six cells in length. This means that the rule still defaults to the all 1s configuration, but if there is a sufficiently large island of W in a low-ρ IC (a fairly likely event), the W island expands and the IC is correctly classified. However, misclassification is now possible for $\rho_0 > 0.5$ due to the (less likely) chance formation of an island of W of length six. This aspect of the strategy is similar to the "block-expanding" strategy.

In addition to the above changes, a new interaction is allowed to take place for the first time. The decrease in velocity of the particle P_{BW} to 0.5 not only results in the removal of small islands of B but it also allows $P_{\#B}$ to interact with P_{BW}. The interaction results in the particle $P_{\#W}$ with a velocity of -3.0, creating the necessary symmetry with $P_{B\#}$, which has velocity +3.0. From this juncture the # domain and its four associated particles play a major role in determining the fitness of a CA rule.

After a brief stasis over three generations, two developments result in the improved performance seen in ϕ_{2e}, the best rule in generation 16 ($F_{100} = 0.89$). First, P_{WB} now spontaneously creates a new # domain with two boundaries, one on each side: $P_{W\#}$ moving to the left with a velocity of -1.0, and $P_{\#B}$ moving to the right with a velocity of 1.0 (Figure 2(e)). Since $P_{W\#}$ is moving to the left and can interact with right moving P_{BW} to create $P_{B\#}$, the rule will stop the growth of W domains of length more than 6 cells when surrounded by larger regions of B. This prevents the "block-expanding strategy" error of expanding W blocks when $\rho_0 > 0.5$. This is a major innovation in using particles to effect non-local computation.

Second, the velocity of P_{BW} is further reduced from 0.5 to reduced to 1/3. However, an asymmetry still remains: because P_{BW} is moving to the right with a positive velocity, it takes longer to remove islands of B than to remove islands of W. This asymmetry is rectified by the discovery of ϕ_{2f} in generation 18, in which the velocity of P_{BW} is set to zero (Figure 2(f)), yielding $F_{100} = 0.98$. From generation 19 till the end of the GA run, little change is seen in either the fitness of the best rule in the population or its behavior. ϕ_{2f}'s behavior is almost identical to that of ϕ_{1d} which was discovered later in this run.

Interestingly, ϕ_{1d}'s behavior is very similar to the behavior of the well-known Gacs-Kurdyumov-Levin (GKL) CA, which was invented to study reliable computation and phase transitions in one-dimensional spatially-extended systems [6]. As shown in Table 1, ϕ_{GKL} has higher $\mathcal{P}^N_{10^4}$ than any rule found by the GA, although ϕ_{GKL} and ϕ_{1d} produce the same three domains $B, W, \#$, and also result in identical particles, particle velocities, and particle interactions. The difference in performance results, for example, from asymmetries in ϕ_{1d} not present in ϕ_{GKL} (see [9]); the detailed differences will be described in future work.

5. Conclusion

The results presented here demonstrate how evolution can engender emergent computation in a spatially-extended system. The GA found CAs that used particle-based computation to achieve performance nearly as high as that of the best human-designed rule on a density classification task. These discoveries are encouraging, both for using GAs to model the evolution of emergent computation in nature and for the automatic engineering of emergent computation in decentralized multicomponent systems. In previous work, a number of factors have been identified that limited the GA's performance in our experiments, and suggestions have been made for overcoming these impediments [9]. Even with these impediments, our GA was able to discover sophisticated CAs reliably, though with a low frequency. We expect that these approaches will eventually result in evolutionary-computation methods that discover sophisticated particle-based computation for a wide variety of applications. For example, in work that will be reported elsewhere, the GA discovered CAs that used particle-based computation to rapidly achieve stable global synchronization between local processors.

Acknowledgements. This research was supported by the Santa Fe Institute, under the Adaptive Computation and External Faculty Programs, by NSF grant IRI-9320200, and by the University of California, Berkeley, under contract AFOSR 91-0293.

References

[1] J. P. Crutchfield. The calculi of emergence: Computation, dynamics, and induction. *Physica D*, in press.

[2] J. P. Crutchfield (1994). Is anything ever new? Considering emergence. In G. Cowan, D. Pines, and D. Melzner (editors), *Complexity: Metaphors, Models, and Reality*, 479-497. Reading, MA: Addison-Wesley.

[3] J. P. Crutchfield and J. E. Hanson (1993). Turbulent Pattern Bases for Cellular Automata. *Physica D*, 69:279-301.

[4] J. P. Crutchfield and M. Mitchell (1994). The Evolution of Emergent Computation. Working Paper 94-03-012, Santa Fe Institute, Santa Fe, New Mexico.

[5] S. Forrest (1990). Emergent Computation: Self-organizing, collective, and cooperative behavior in natural and artificial computing networks: Introduction to the Proceedings of the Ninth Annual CNLS Conference. *Physica D*, 42:1–11.

[6] P. Gacs (1985). Nonergodic one-dimensional media and reliable computation. *Contemporary Mathematics*, 41:125.

[7] J. E. Hanson and J. P. Crutchfield (1992). The attractor-basin portrait of a cellular automaton. *Journal of Statistical Physics*, 66(5/6): 1415–1462.

[8] C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen (editors) (1992). *Artificial Life II*. Reading, MA: Addison-Wesley.

[9] M. Mitchell, J. P. Crutchfield, and P. T. Hraber. Evolving cellular automata to perform computations: Mechanisms and impediments. *Physica D*, in press.

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

[11] N. H. Packard (1988). Adaptation toward the edge of chaos. In J. A. S. Kelso, A. J. Mandell, and M. F. Shlesinger (editors), *Dynamic Patterns in Complex Systems*, 293–301. Singapore: World Scientific.

[12] T. Toffoli and N. Margolus (1987). *Cellular automata machines: A new environment for modeling*. Cambridge, MA: MIT Press.

[13] S. Wolfram (1986). *Theory and applications of cellular automata*. Singapore: World Scientific.