Evolving Cellular Automata with Genetic Algorithms: A Review of Recent Work

Melanie Mitchell
Santa Fe Institute
1399 Hyde Park Road
Santa Fe, NM 87501
mm@santafe.edu

James P. Crutchfield[1]
Santa Fe Institute
1399 Hyde Park Road
Santa Fe, NM 87501
jpc@santafe.edu

Rajarshi Das
IBM Watson Research Ctr.
P.O. Box 704
Yorktown Heights, NY 10598
rajarshi@watson.ibm.com

To appear in *Proceedings of the First International Conference on Evolutionary Computation and Its Applications (EvCA'96).*
Moscow, Russia: Russian Academy of Sciences.

Abstract

We review recent work done by our group on applying genetic algorithms (GAs) to the design of cellular automata (CAs) that can perform computations requiring global coordination. A GA was used to evolve CAs for two computational tasks: density classification and synchronization. In both cases, the GA discovered rules that gave rise to sophisticated emergent computational strategies. These strategies can be analyzed using a "computational mechanics" framework in which "particles" carry information and interactions between particles effects information processing. This framework can also be used to explain the process by which the strategies were designed by the GA. The work described here is a first step in employing GAs to engineer useful emergent computation in decentralized multi-processor systems. It is also a first step in understanding how an evolutionary process can produce complex systems with sophisticated collective computational abilities.

Introduction

In our work we are studying how genetic algorithms (GAs) can evolve cellular automata (CAs) to perform computations that require global coordination. The "evolving cellular automata" framework is an idealized means for studying how evolution (natural or computational) can create systems in which "emergent computation" takes place—that is, in which the actions of simple components with local information and communication 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 such emergent computation occurs. However, it is not well understood *how* these natural systems perform computations. Our ultimate motivations are both to understand emergent computation in natural systems and to explore ways of engineering sophisticated emergent computation in decentralized multi-processor systems. Previous papers on this topic include Mitchell, Hraber, and Crutchfield 1993; Mitchell, Crutchfield, and Hraber 1994; Das,

[1]also Department of Physics, University of California, Berkeley, CA 90720-7300

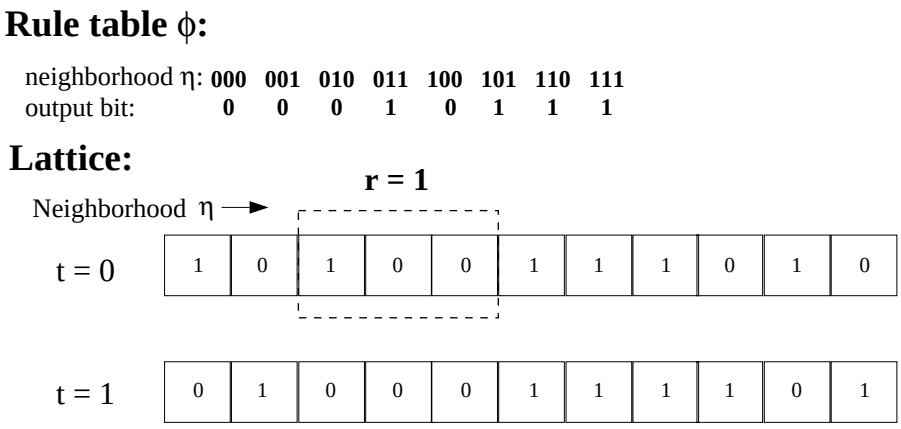

Figure 1: Illustration of a one-dimensional, binary-state, nearest-neighbor ($r = 1$) cellular automaton with $N = 11$. Both the lattice and the rule table ϕ for updating the lattice are illustrated. The lattice configuration is shown at two successive time steps. The cellular automaton has spatially periodic boundary conditions: the lattice is viewed as a circle, with the leftmost cell being the right neighbor of the rightmost cell, and vice versa.

Mitchell, Crutchfield 1994; Crutchfield and Mitchell 1995; Das, Crutchfield, Mitchell, and Hanson, 1995, and Mitchell, Crutchfield, and Das, 1996. (These papers can be obtained on the World Wide Web URL http://www.santafe.edu/projects/evca). Here we review work that was first presented in these papers.

Cellular Automata

In this paper we describe work on one-dimensional binary-state cellular automata (CAs). Such a CA consists of 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 well-known "game of Life," Berlekamp, Conway, and Guy 1982, is an example of a two-dimensional CA.) As is illustrated in figure 1, the lattice starts out with an initial configuration (IC) of cell states (0s and 1s) and this configuration changes in discrete time steps in which all cells are updated simultaneously according to the CA "rule" ϕ. (Here we use the term "state" to refer to the value of a single cell. The term "configuration" will refer to the collection of local states over the entire lattice.)

A CA's rule ϕ can be expressed as a lookup table ("rule table," or "CA rule") that lists, for each local neighborhood, the state which is taken on by the neighborhood's central cell at the next time step. For a binary-state CA, these update states are referred to as the "output bits" of the rule table. In a one-dimensional CA, a neighborhood consists of a cell and its r ("radius") neighbors on either side. (In figure 1, $r = 1$.) Here we describe CAs with periodic boundary conditions—the lattice is viewed as a circle.

Cellular automata have been studied extensively as mathematical objects, as models of natural systems, and as architectures for fast, reliable parallel computation (e.g., see Wolfram 1986 for an overview of these various roles). One-dimensional binary-state cellular automata

are perhaps the simplest examples of decentralized, spatially extended systems in which emergent computation can be studied. In our project, a CA performing a computation means that the input to the computation is encoded as the IC, the output is decoded from the configuration reached at some later time step, and the intermediate steps that transform the input to the output are taken as the steps in the computation. The computation emerges from the CA rule being obeyed by each cell. (Note that this use of CAs as computers differs from the impractical, though theoretically interesting, method of constructing a universal Turing machine in a CA; see Mitchell, Hraber, and Crutchfield 1993 for a comparison of these two approaches.)

Computational Tasks for Cellular Automata

Some early work on evolving CAs with GAs was done by Packard and colleagues (Packard 1988; Richards, Meyer, and Packard 1992). Koza (1992) also applied genetic programming to evolve CAs for simple random-number generation.

Our work builds on that of Packard (1988). In preliminary work, we have used a form of the GA to evolve one-dimensional, binary-state $r = 3$ CAs to perform a density-classification task (Crutchfield and Mitchell 1995; Das, Mitchell, and Crutchfield 1994) and a synchronization task (Das, Crutchfield, Mitchell, and Hanson 1995).

For the density classification task, the goal was to find a CA that decides whether or not the IC contains a majority of 1s (i.e., has high density). More precisely, we call this task the "$\rho_c = 1/2$" task. Here ρ denotes the density of 1s in a binary-state CA configuration and ρ_c denotes a "critical" or threshold density for classification. Let ρ_0 denote the density of 1s in the IC. If $\rho_0 > \rho_c$, then within M time steps the CA should go to the fixed-point configuration of all 1s (i.e., all cells in state 1 for all subsequent iterations); otherwise, within M time steps it should produce the fixed-point configuration of all 0s. M is a parameter of the task that depends on the lattice size N.

Designing an algorithm to perform the $\rho_c = 1/2$ task is trivial for a system with a central controller or central storage of some kind, such as a standard computer with a counter register or a neural network in which all input units are connected to a central hidden unit. However, it is nontrivial to design a a small-radius ($r \ll N$) CA to perform this task, since a small-radius CA relies only on local interactions. It has been argued that no finite-radius, binary CA with periodic boundary conditions can perform this task perfectly across all lattice sizes (Land and Belew, 1995; Das, 1996), 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. Since the 1s can be distributed throughout the CA lattice, the CA must transfer information over large distances ($\approx N$), and process information collected from different parts of the lattice. To do this requires the global coordination of cells that are separated by large distances and that cannot communicate directly.

The need for such coordination is illustrated in figure 2, in which we display the space-time behavior of a "naive" hand-designed candidate solution for this task—the "majority" rule ϕ_{maj}, in which the output bit for each 7-bit ($r = 3$) neighborhood is decided by a majority

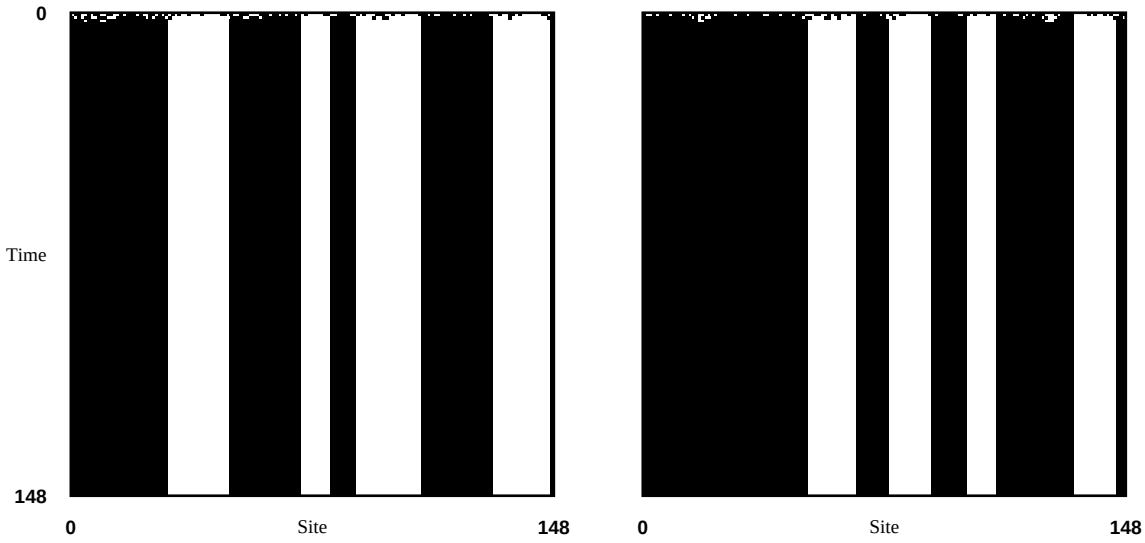

Figure 2: Space-time diagrams for ϕ_{maj}, the $r = 3$ majority rule. In the left diagram, $\rho_0 < \frac{1}{2}$; in the right diagram, $\rho_0 > \frac{1}{2}$.

vote among the seven cells in the neighborhood. Figure 2 gives two "space-time diagrams," displaying the behavior of this rule on two initial conditions, one with $\rho_0 < 1/2$ and the other with $\rho_0 > 1/2$. Here, lattice configurations are plotted over a series of time steps, with 1s given as black cells and 0s given as white cells, and with time increasing down the page. As can be seen, local neighborhoods with majority 1s map to regions of all 1s and similarly for 0s, but when an all-1s region and an all-0s region border each other, there is no way to decide between them, and both persist. Thus, the majority rule (which implements a threshold on a linear combination of states) does not perform the $\rho_c = 1/2$ task.

Instead, more sophisticated coordination and information transfer must be achieved. This coordination must, of course, happen in the absence of any central processor or central memory directing the coordination.

Evolving Cellular Automata with Genetic Algorithms

We used a genetic algorithm to search for $r = 3$ CA rule tables to perform the $\rho_c = 1/2$ task. Each chromosome in the population represented a candidate rule table—it consisted of the output bits of the rule table, listed in lexicographic order of neighborhood (cf. ϕ in figure 1). The chromosomes representing rules were thus of length $2^{2r+1} = 128$. The size of the rule space in which the GA worked was thus 2^{128}—far too large for any kind of exhaustive evaluation.

In our main set of experiments, we set $N = 149$, a reasonably large but still computationally tractable odd number (odd, so that the task will be well-defined on all ICs). The GA began with a population of 100 chromosomes chosen at random from a distribution that was flat over the density of 1s in the output bits. (This "uniform" distribution differs from the more commonly used "unbiased" distribution in which each bit in the rule table is indepen-

dently randomly chosen. We found that using a uniform distribution considerably improved the GA's performance on this task—see Mitchell, Crutchfield, and Hraber 1994, for details). The fitness of a rule in the population was computed by (1) randomly choosing 100 ICs that are uniformly distributed over $\rho \in [0.0, 1.0]$, with exactly half with $\rho < \rho_c$ and half with $\rho > \rho_c$, (2) iterating the CA on each IC until it arrives at a fixed point or for a maximum of $M \approx 2N$ time steps, and (3) determining whether the final behavior is correct—i.e., 149 0s for $\rho_0 < \rho_c$ and 149 1s for $\rho_0 > \rho_c$. The rule's fitness, F_{100}, was the fraction of the 100 ICs on which the rule produced the correct final behavior. No partial credit was given for partially correct final configurations.

A few comments about the fitness function are in order. First, the number of possible input cases (2^{149} for $N = 149$) was far too large for fitness to be defined as the fraction of correct classifications over all possible ICs. Instead, fitness was defined as the fraction of correct classifications over a sample consisting of 100 ICs. A different sample was chosen at each generation, making the fitness function stochastic. In addition, like the initial CA population, the ICs were not sampled from an unbiased distribution (i.e., equal probability of a 1 or a 0 at each site in the IC), but rather from a flat ("uniform") distribution across $\rho \in [0, 1]$ (i.e., ICs of each density from $\rho = 0$ to $\rho = 1$ were approximately equally represented). This uniform distribution was used because the unbiased distribution is binomially distributed and thus very strongly peaked at $\rho = 1/2$. The ICs selected from such a distribution will likely all have $\rho \approx 1/2$, the hardest cases to classify. In experiments using an unbiased IC sample to calculate fitness, the GA was very rarely able to discover high-fitness CAs.

Our version of the GA worked as follows. In each generation, (1) a new set of 100 ICs was generated, (2) F_{100} was computed on this set for each rule in the population, (3) CAs in the population were ranked in order of fitness, (4) the 20 highest fitness ("elite") rules were copied to the next generation without modification, and (5) the remaining 80 rules for the next generation were formed by single-point crossovers between randomly chosen pairs of elite rules. The parent rules were chosen from the elite with replacement—that is, an elite rule was permitted to be chosen any number of times. The offspring from each crossover were each mutated at exactly two randomly chosen positions. This process was repeated for 100 generations for a single run of the GA.

Our selection scheme, in which the top 20% of the rules in the population are copied without modification to the next generation and the bottom 80% are replaced, is similar to the $(\mu + \lambda)$ selection method used in some evolution strategies (see Bäck, Hoffmeister, and Schwefel 1991). Selecting parents by relative fitness rank rather than in proportion to absolute fitness helps to prevent initially stronger individuals from too quickly dominating the population and driving the genetic diversity down early. Also, since testing a rule on 100 ICs provides only an approximate gauge of the rule's performance over all 2^{149} possible ICs, saving the top 20% of the rules is a good way of making a "first cut" and allowing rules that survive to be tested over different ICs in succeeding generations. 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.

CA ($r = 3$)	Rule table (hex)	$\mathcal{P}_{149,10^4}$	$\mathcal{P}_{599,10^4}$	$\mathcal{P}_{999,10^4}$
$\phi_{\rm maj}$	000101170117177f 0117177f177f7fff	0.000	0.000	0.000
$\phi_{\rm exp}$	0505408305c90101 200b0efb94c7cff7	0.652	0.515	0.503
$\phi_{\rm par}$	0504058705000f77 037755837bffb77f	0.769	0.725	0.714

Table 1: Rule tables and measured values of $\mathcal{P}_{N,10^4}(\phi)$ at various N for three different $r = 3$ rules. To recover the 128-bit string giving the CA look-up table output bits, expand each hexadecimal digit (the first row followed by the second row) to binary. The output bits are then given in lexicographic order starting from the all-0s neighborhood at the leftmost bit in the 128-bit binary string. $\phi_{\rm maj}$ (hand-designed) computes the majority of 1s in the neighborhood. $\phi_{\rm exp}$ (evolved by the GA) expands blocks of 1s. $\phi_{\rm par}$ (evolved by the GA) uses a "particle-based" strategy.

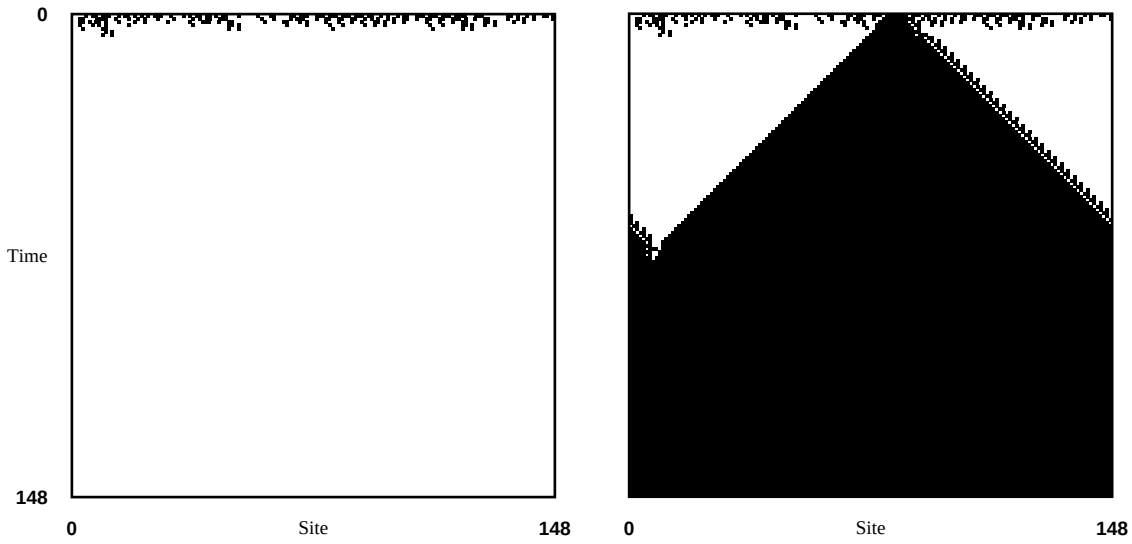

Figure 3: Space-time diagrams for a "block-expanding" rule, $\phi_{\rm exp}$. In the left diagram, $\rho_0 < 1/2$; in the right diagram, $\rho_0 > 1/2$. Both ICs are correctly classified.

Results

In our initial experiments, three hundred different runs were performed, each starting with a different random-number seed. We examined the fittest evolved rules to understand their computational "strategies" for performing the density classification task. On most runs the GA evolved a rather unsophisticated class of strategies. One example, a CA here called $\phi_{\rm exp}$ (for "expand"), is illustrated in figure 3. This rule had $F_{100} \approx 0.9$ in the generation in which it was discovered. Its computational strategy is the following: Quickly reach the 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. (For this rule, "sufficiently large" is 7 or more cells.) This strategy does a fairly good job of classifying low and high density ICs under F_{100}: it relies on the appearance or absence of blocks of 1s to be good predictors of ρ_0, since high-density ICs are statistically more likely to have blocks of adjacent 1s than low-density ICs.

Similar strategies were evolved in most runs. On approximately half the runs, "expand 1s" strategies were evolved, and on most of the other runs, the opposite "expand 0s" strategies were evolved. These block-expanding strategies, although successful given F_{100} and $N = 149$, do not count as sophisticated examples of emergent computation in CAs: all the computation is done locally in identifying and then expanding a "sufficiently large" block. There is no interesting notion of global coordination or information flow between distant cells—two things we claimed were necessary to perform well on the task. Indeed, such strategies perform poorly under performance measures using different distributions of ICs, and when N is increased.

Mitchell, Crutchfield, and Hraber (1994) analyzed the detailed mechanisms by which the GA evolved such block-expanding strategies. This analysis uncovered some quite interesting aspects of the GA, including a number of impediments that, on most runs, kept the GA from discovering better-performing CAs. These included the GA's breaking the $\rho_c = 1/2$ task's symmetries for short-term gains in fitness, as well as "overfitting" to the fixed lattice size $N = 149$ and the unchallenging nature of the IC samples. The last point merits some elaboration here.

The uniform distribution of ICs over $\rho \in [0, 1]$ helped the GA get a leg up in the early generations. We found that computing fitness using an unbiased distribution of ICs made the problem too difficult for the GA early on—it was rarely able to find improvements to the CAs in the initial population. However, the biased distribution became too easy for the improved CAs later in a run (i.e., the low and high density ICs were very easily classified); these ICs did not push the GA hard enough to find better solutions. We are currently exploring a "coevolution" scheme, in which the IC sample is itself subject to selection and variation by the GA (cf. Hillis, 1990), in order to improve the GA's performance on this problem.

Despite these various impediments and the unsophisticated CAs evolved on most runs, on several different runs in our initial experiment the GA discovered CAs with more sophisticated strategies that yielded significantly better performance across different IC distributions and lattice sizes than was achieved by block-expanding strategies. The typical space-time behaviors of one such rule, here called ϕ_{par} (for "particle"), are illustrated in figure 4.

The improved performance of ϕ_{par} can be seen in table 1, which gives the rule tables and performances across different lattice sizes for different rules. The "performance" $\mathcal{P}_{N,10^4}$ is defined as the fraction of correct classifications (N 0s for $\rho_0 < \rho_c$ and N 1s for $\rho_0 > \rho_c$) over 10^4 ICs chosen at random from the unbiased distribution (each bit in the IC is independently randomly chosen). As was mentioned above, these ICs all have ρ_0 close to $1/2$ and are thus the hardest cases to classify; therefore, $\mathcal{P}_{N,10^4}$ gives a lower bound on other performance measures. CA ϕ_{par} not only has significantly higher performance than ϕ_{exp} for $N = 149$, but its performance degrades relatively slowly as N is increased, whereas ϕ_{exp}'s performance drops quickly. As we describe in Das, Mitchell, and Crutchfield 1994, ϕ_{par}'s behavior is similar to that of a CA designed by Gacs, Kurdyumov, and Levin (1978).

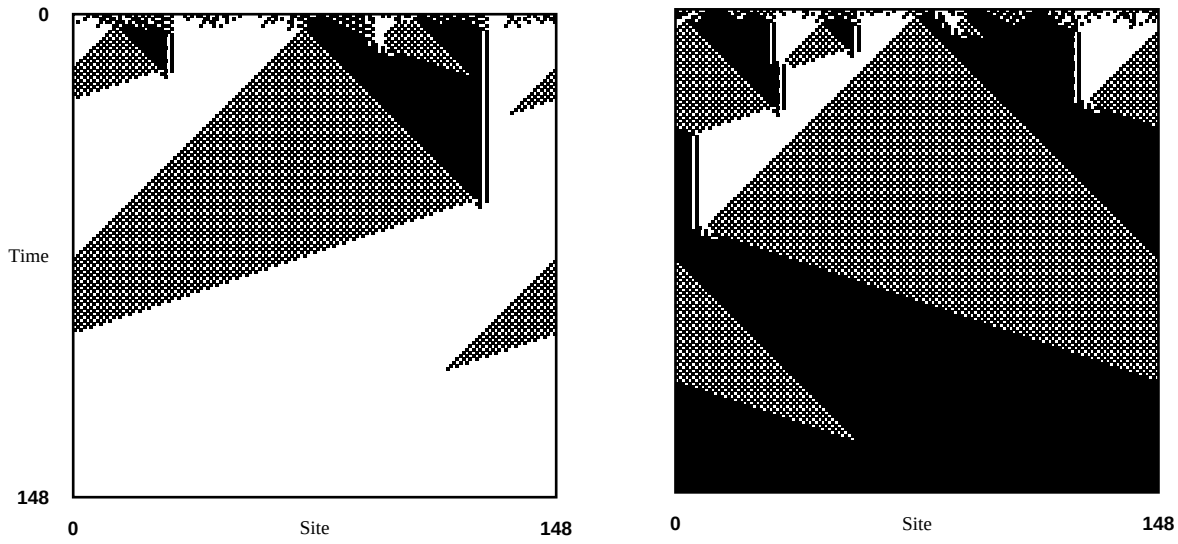

Figure 4: Space-time diagrams for $\phi_{\rm par}$, a "particle-based" rule. In the left diagram, $\rho_0 < 1/2$; in the right diagram, $\rho_0 > 1/2$. Both ICs are correctly classified.

Analysis of Evolved CAs

In figure 4 it can be seen that, under $\phi_{\rm par}$, there is a transient phase during which spatial and temporal transfer of information about the density in local regions takes place. Roughly, over short times, $\phi_{\rm par}$'s behavior is locally similar to that of $\phi_{\rm maj}$ in that local high-density regions are mapped to all 1s, local low-density regions are mapped to all 0s, with a vertical boundary in between them. This is what happens when a region of 1s on the left meets a region of 0s on the right. However, there is a crucial difference from $\phi_{\rm maj}$: when a region of 0s on the left meets a region of 1s on the right, rather than a vertical boundary being formed, a checkerboard region (alternating 1s and 0s) is propagated. When the propagating checkerboard region collides with the black-white boundary, the inner region (e.g., each of the white regions in the right-hand diagram of figure 4) is cut off and the outer region is allowed to propagate. In this way, the CA uses local interactions and geometry to determine the relative sizes of adjacent large low and high density regions. For example, in the right-hand space-time diagram, the large inner white region is smaller than the large outer black region—thus the propagating checkerboard pattern reaches the black-white boundary on the white side before it reaches it on the black side; the former is cut off, and the latter is allowed to propagate.

The black-white boundary and the checkerboard region can be thought of as "signals" indicating "ambiguous" regions. The creation and interactions of these signals can be interpreted as the locus of the computation being performed by the CA—they form its emergent "algorithm."

The above explanation of how $\phi_{\rm par}$ performs the $\rho_c = 1/2$ task is informal and incomplete. Can we understand more rigorously how the evolved CAs perform the desired computation? Understanding the products of GA evolution is a general problem—typically the GA is asked

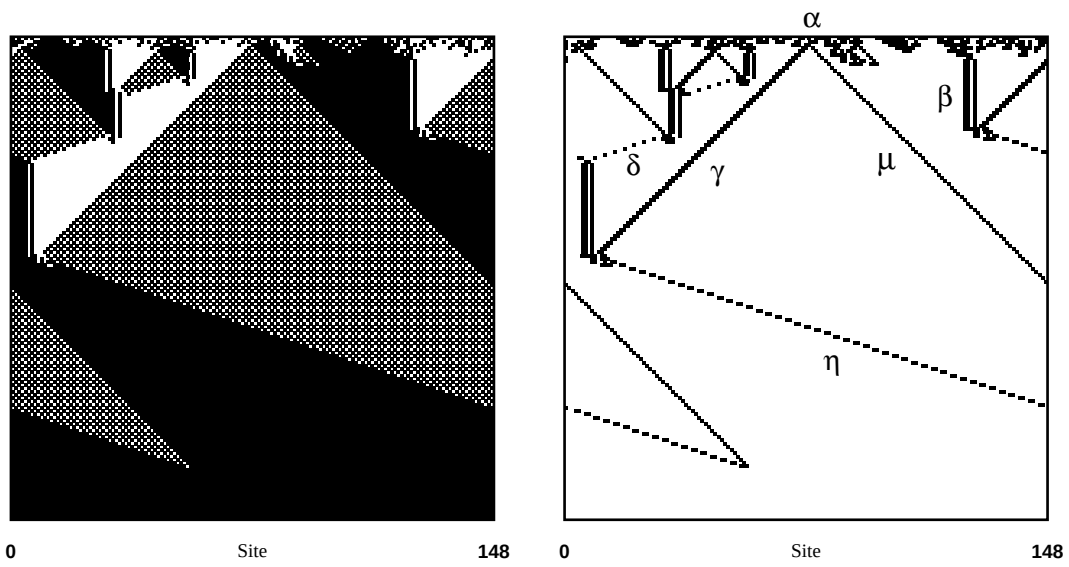

Figure 5: (a) The right-hand spacetime diagram of figure 4. (b) The same diagram with the regular domains filtered out, leaving only the particles (some of which are labeled by here by the Greek letter code of table 2). Note that particle α (unlike other the other particles) lasts for only one time step, after which it decays to particles γ and μ.

to find individuals that achieve high fitness but is not told what traits the individuals should have to attain high fitness. This is analogous to the difficulty biologists have in understanding the products of natural evolution. In many cases, particularly in automatic-programming applications (e.g., genetic programming, Koza 1992), it is difficult to understand exactly how an evolved high-fitness individual works. The problem is compounded in the case of cellular automata, since the emergent computation performed by a given CA is determined by its overall space-time behavior, and is thus almost always impossible to extract from the bits of the rule table.

A more promising approach is to examine the space-time behavior exhibited by the CA and to "reconstruct" from that behavior what the emergent algorithm is. Crutchfield and Hanson have developed a general method for reconstructing the "intrinsic" computation embedded in space-time behavior in terms of "regular domains," "particles," and "particle interactions" (Hanson and Crutchfield, 1992; Crutchfield and Hanson 1993). This method is part of their "computational mechanics" framework for understanding computation in physical systems (Crutchfield, 1994). A detailed discussion of computational mechanics and particle-based computation is beyond the scope of this paper. Very briefly, regular domains are regions of space-time consisting of words in the same regular language—in other words, they are regions that are computationally homogeneous and simple to describe. E.g., in figure 4, there are three regular domains, corresponding to the regular languages 0^*, 1^*, and $(01)^*$. Particles are the localized boundaries between those domains. In computational mechanics, particles are identified as information carriers, and collisions between particles are identified as the loci of information processing. Particles and particle interactions form a high-level language for describing computation in spatially extended systems such as CAs. Figure 5 hints at this higher level of description: to produce it we filtered the regular domains

Regular Domains		
$\Lambda^0 = 0^*$	$\Lambda^1 = 1^*$	$\Lambda^2 = (01)^*$

Particles (Velocities)	
$\alpha \sim \Lambda^0\Lambda^1$ (0)	$\beta \sim \Lambda^1 01 \Lambda^0$ (0)
$\gamma \sim \Lambda^0\Lambda^2$ (-1)	$\delta \sim \Lambda^2\Lambda^0$ (-3)
$\eta \sim \Lambda^1\Lambda^2$ (3)	$\mu \sim \Lambda^2\Lambda^1$ (1)

Interactions	
decay	$\alpha \rightarrow \gamma + \mu$
react	$\beta + \gamma \rightarrow \eta$, $\mu + \beta \rightarrow \delta$, $\eta + \delta \rightarrow \beta$
annihilate	$\eta + \mu \rightarrow \emptyset_1$, $\gamma + \delta \rightarrow \emptyset_0$

Table 2: Catalog of regular domains, particles (domain boundaries), particle velocities (in parentheses), and particle interactions seen in $\phi_{\rm par}$'s space-time behavior. The notation $p \sim \Lambda^x\Lambda^y$ means that p is the particle forming the boundary between regular domains Λ^x and Λ^y.

from the space-time behavior displayed in the right-hand diagram of figure 4 to leave only the particles and their interactions, in terms of which the emergent algorithm of the CA can be understood. Table 2 gives a catalog of the relevant particles and interactions for $\phi_{\rm par}$. In Crutchfield and Mitchell (1995) and in Das (1996) we describe other particle-based rules that were evolved by the GA for this task.

The application of computational mechanics to the understanding of rules evolved by the GA is discussed further in Das, Mitchell, and Crutchfield 1994, in Das, Crutchfield, Mitchell, and Hanson 1995, and in Crutchfield and Mitchell 1995. In the first two papers, we used particles and particle interactions to describe the evolutionary stages in which highly fit rules were evolved by the GA. Figure 6 illustrates these stages ("epochs") for the GA run leading to $\phi_{\rm par}$. Figure 6a plots the best fitness in the population over 50 generations, and labels those generations in which a significantly improved new strategy is discovered. The other plots give space-time diagrams illustrating each of these strategies.

In generation 0 (Figure 6b), the best strategy is: "From any IC, immediately reach a fixed point of all 1s." This trivial solution correctly classifies half the IC sample (the half with $\rho_0 > 1/2$), yielding fitness 0.5. At generation 8, a CA with fitness 0.61 is discovered. This CA quickly reaches the all-0s fixed point for very low-density ICs. For all other ICs (e.g., the density 0.25 IC shown in the figure) it expands very small regions of 1s to reach the all-1s fixed point. The checkerboard pattern is produced when a white region on the left meets a black region on the right, but in this CA the checkerboard pattern is "adaptively neutral"—it does not contribute to the increase in fitness. In particular, if the rule table of this CA is modified so that the checkerboard pattern is no longer produced, the fitness does not change. However, in later generations, by changing bits that affect the velocities of particles, the GA shapes the checkerboard pattern so that it performs a useful function. This type of phenomenon, in which a trait starts out being adaptively neutral but is later shaped by evolution to have adaptive value, is known as "exaptation," and is hypothesized by some evolutionary biologists to be an important component of evolution (e.g., see Gould and Vrba, 1982).

In generations 13–18 (figure 6c–f),the $\phi_{\rm par}$ strategy described above is approached with increasingly better approximations. For example, in generation 16, the particle forming the

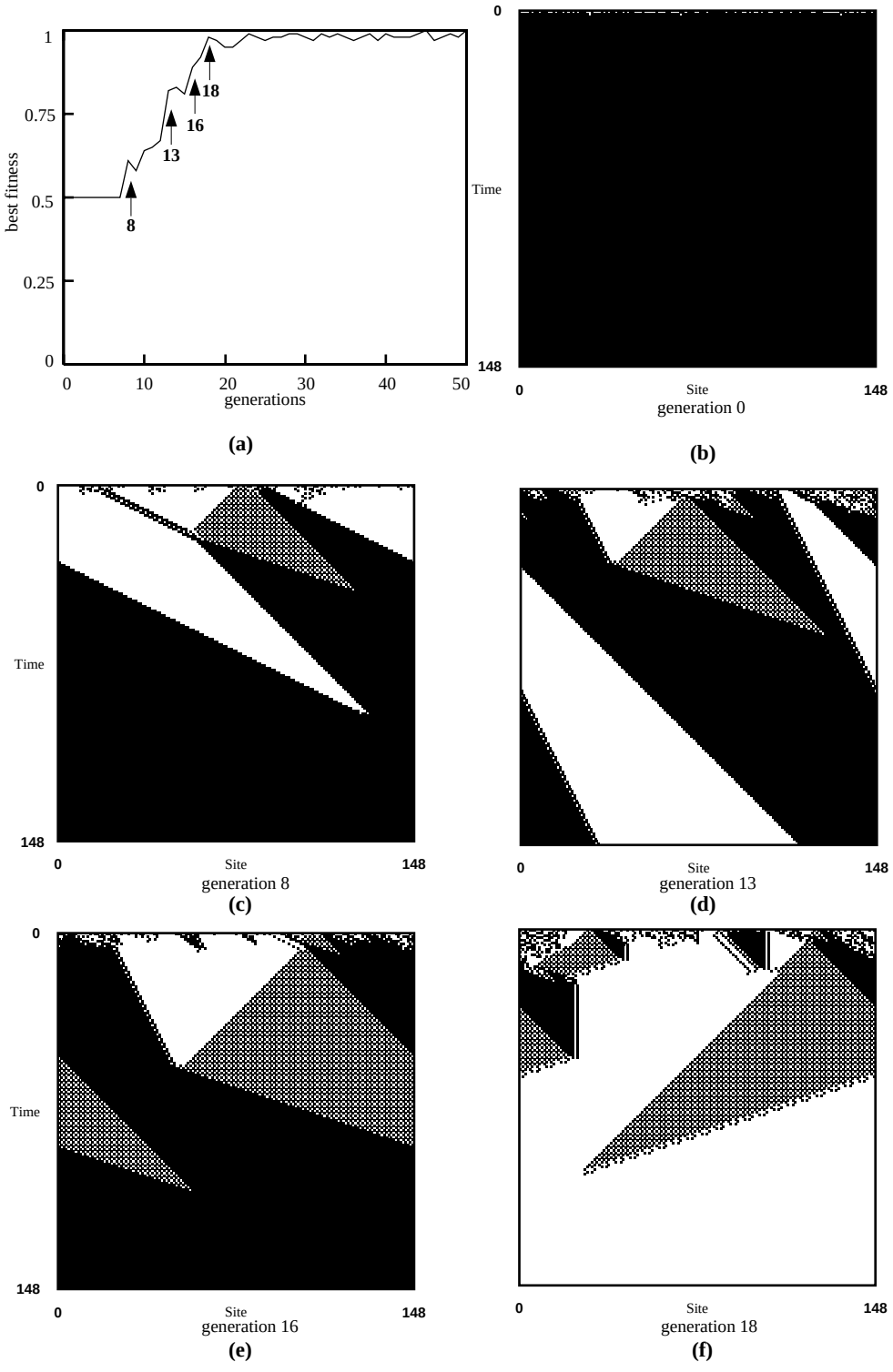

Figure 6: A partial evolutionary history of the GA run leading to ϕ_{par}: (a) F_{100} versus generation for the most fit CA in each population. The arrows indicate the generations in which the GA discovered each new significantly improved strategy. (b)–(f) Space-time diagrams illustrating the behavior of the best ϕ at each of the five generations marked in (a). The ICs were chosen to illustrate particular space-time behaviors.

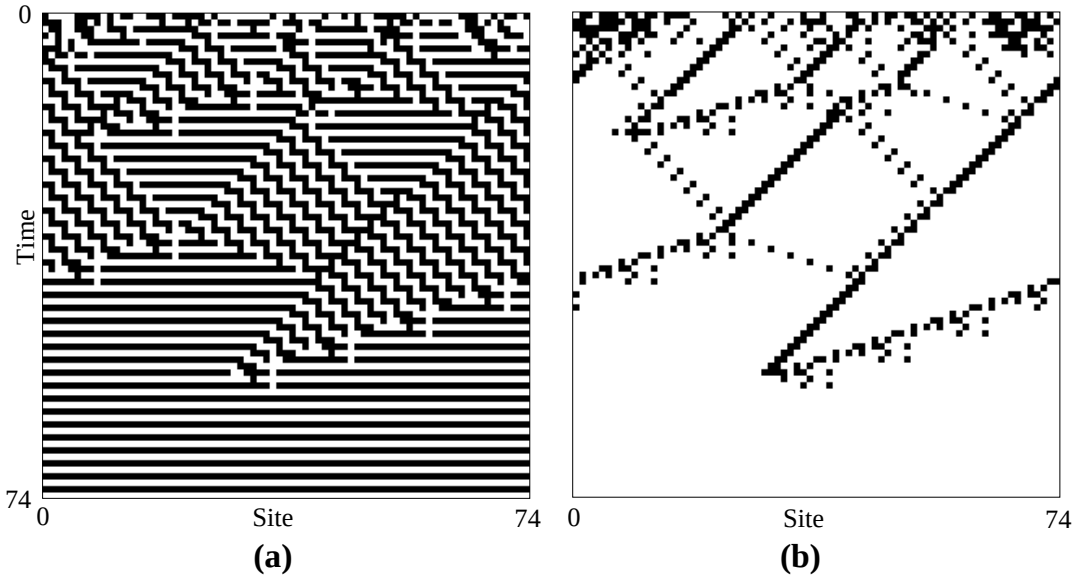

Figure 7: (a) Space-time diagram illustrating the behavior of a CA evolved by the GA to perform the synchronization task. The initial condition was generated at random. (b) The same space-time diagram after filtering out regular domains.

black-white boundary has velocity 1/2, which means that in some cases the white region will get cut off before the black region even when the white region is larger (the expanding black region will encroach on it). In generation 18 the GA has corrected this problem—the black-white boundary has zero velocity, yielding a higher fitness. (A more detailed analysis of these evolutionary epochs is given in Das, Mitchell, and Crutchfield 1994 and in Das, 1996.)

In Das, Crutchfield, Mitchell, and Hanson, 1995, we described a similar analysis for a global synchronization task. The goal for the GA was to find a CA that, from any IC, produces a globally synchronous oscillation between the all-1s and all-0s configurations. (This is perhaps the simplest version of the emergence of spontaneous synchronization that occurs in decentralized systems throughout nature.) Figure 7 illustrates the behavior of one CA evolved by the GA to perform this task, given both as a space-time diagram and as a filtered version of that diagram which reveals the embedded particles. Again, tools of computational mechanics allowed us to understand the computational strategy of this CA in in the higher-level language of particles and particle interactions as opposed to the low-level language of CA rule tables and raw spatial configurations.

Conclusions

The GA's discoveries of rules such as ϕ_{par} and of rules that produce global synchronization is significant, since these are the first examples of a GA's producing sophisticated emergent computation in decentralized, distributed systems such as CAs. These discoveries are encouraging for the prospect of using GAs to automatically evolve computation for more complex

tasks (e.g., image processing or image compression) and in more complex systems; these are the subjects of current work by our group. Moreover, evolving CAs with GAs also gives us a tractable framework in which to study the mechanisms by which an evolutionary process might create complex coordinated behavior in natural decentralized distributed systems. For example, we have already learned how the GA's breaking of symmetries can lead to suboptimal computational strategies (Mitchell, Crutchfield, and Hraber 1993); eventually we may be able to use such models to test ways in which such symmetry breaking might occur in natural evolution. In general, models such as ours can provide insights on how evolutionary processes can discover structural properties of individuals that give rise to improved adaptation. In our case, such structural properties—regular domains and particles—were identified via the computational mechanics framework, and allowed us to analyze the evolutionary emergence of sophisticated computation.

Acknowledgments

This research was supported by the Santa Fe Institute, under the Adaptive Computation and External Faculty Programs and under NSF grant IRI-9320200 and DOE grant DE-FG03-94ER25231. It was supported by the University of California, Berkeley, under the ONR Dynamical Neural Systems Program and AFOSR grant 91-0293.

References

Bäck, T., Hoffmeister, F., and Schwefel, H.-P. 1991. A survey of evolution strategies. In R. K. Belew and L. B. Booker, eds., *Proceedings of the Fourth International Conference on Genetic Algorithms*, 2–9. San Francisco, CA: Morgan Kaufmann.

Berlekamp, E., Conway, J. H., and Guy, R. 1982. *Winning Ways for Your Mathematical Plays*, volume 2. New York: Academic Press.

Crutchfield, J. P. 1994. The calculi of emergence: Computation, dynamics, and induction. *Physica D* 75: 11-54.

Crutchfield, J. P., and Hanson, J. E. 1993. Turbulent pattern bases for cellular automata. *Physica D* 69: 279–301.

Crutchfield, J. P., and Mitchell, M. 1995. The evolution of emergent computation. *Proceedings of the National Academy of Sciences, USA*, 92 (23): 10742.

Das, R. 1996. *The Evolution of Emergent Computation in Cellular Automata.* Ph.D. Thesis, Computer Science Department, Colorado State University, Ft. Collins, CO.

Das, R., Crutchfield, J. P., Mitchell, M., and Hanson, J. E. 1995. Evolving globally synchronized cellular automata. In L. J. Eshelman, ed., *Proceedings of the Sixth International Conference on Genetic Algorithms*, 336–343. San Francisco, CA: Morgan Kaufmann.

Das, R., Mitchell, M., and Crutchfield, J. P. 1994. A genetic algorithm discovers particle-based computation in cellular automata. In Y. Davidor, H.-P. Schwefel, and R.

Männer, eds., *Parallel Problem Solving from Nature—PPSN III*, 244-353. Berlin: Springer-Verlag (Lecture Notes in Computer Science, volume 866).

Gacs, P., Kurdyumov, G. L., and Levin, L. A. 1978. One-dimensional uniform arrays that wash out finite islands. *Problemy Peredachi Informatsii* 14: 92–98 (in Russian).

Gould, S. J., and Vrba, E. S., 1982. Exaptation: A missing term in the science of form. *Paleobiology* 8: 4–15.

Hanson, J. E., and Crutchfield, J. P. 1992. The attractor-basin portrait of a cellular automaton. *Journal of Statistical Physics* 66, no. 5/6: 1415–1462.

Hillis, W. D. 1990. Co-evolving parasites improve simulated evolution as an optimization procedure. *Physica D* 42: 228–234.

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

Land, M., and Belew, R. K. 1995. No perfect two-state cellular automata for density classification exists. *Physical Review Letters* 74 (25): 5148.

Mitchell, M., Crutchfield, J. P., and Das, R. (In press). Evolving cellular automata to perform computations. In Bäck, T., Fogel, D., and Michalewicz, Z. (Eds.), *Handbook of Evolutionary Computation.* Oxford: Oxford University Press.

Mitchell, M., Crutchfield, J. P., and Hraber, P. T. 1994. Evolving cellular automata to perform computations: Mechanisms and impediments. *Physica D* 75: 361–391.

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

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

Richards, F. C., Meyer, T. P., and Packard, N. H. 1990. Extracting cellular automaton rules directly from experimental data. *Physica D* 45: 189–202.

Wolfram, S. 1986. *Theory and Applications of Cellular Automata.* Singapore: World Scientific.