G1.15: Computer Science Application: Evolving Cellular Automata to Perform Computations

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[2]
Santa Fe Institute
1399 Hyde Park Road
Santa Fe, NM 87501
raja@santafe.edu

In Bäck, T., Fogel, D., and Michalewicz, Z. (Eds.), *Handbook of Evolutionary Computation.* Oxford: Oxford University Press, 1997.

Abstract

We describe an application of 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 interaction between particles effects information processing. This framework can also be used to explain the process by which the strategies are designed by the GA. This work is a first step in employing GAs to engineer useful emergent computation in decentralized multi-processor systems.

Project overview

An example of automatic programming by genetic algorithms (GAs) is found in our work on evolving cellular automata to perform computations (Mitchell, Hraber, and Crutchfield 1993; Mitchell, Crutchfield, and Hraber 1994; Crutchfield and Mitchell 1994; Das, Mitchell, and Crutchfield 1994; Das, Crutchfield, Mitchell, and Hanson, 1995; these papers can be obtained on the World Wide Web URL http://www.santafe.edu/projects/evca). This project has elements of both engineering and scientific modeling. One motivation is to understand how natural evolution creates 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 (Forrest 1990; Langton 1992). However, it is not well understood *how* these natural systems perform computations. Another motivation is to find ways to engineer sophisticated emergent computation in decentralized multi-processor systems.

[1]also Department of Physics, University of California, Berkeley, CA 90720-7300
[2]also Department of Computer Science, Colorado State University, Fort Collins, CO 80523

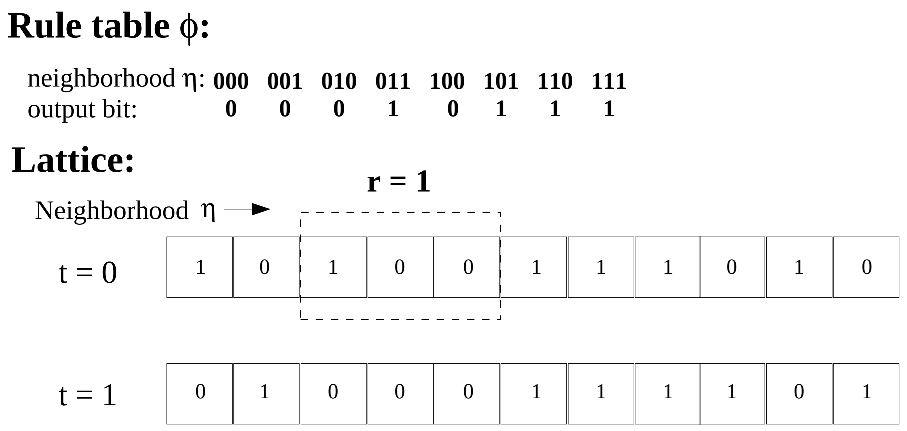

Figure 1: Illustration of a one-dimensional, binary-state, nearest-neighbor ($r = 1$) cellular automaton with $N = 11$. Both the lattice and the rule table ϕ for updating the lattice are illustrated. The lattice configuration is shown over one time step. The cellular automaton has spatially periodic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

One of the simplest decentralized, spatially extended 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 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") 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. (For overviews of CA theory and applications, see Wolfram 1986 and Toffoli and Margolus 1987.) 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 use in science and engineering and for general computation. Here we describe work on using GAs to engineer CAs to perform computations.

Typically, 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

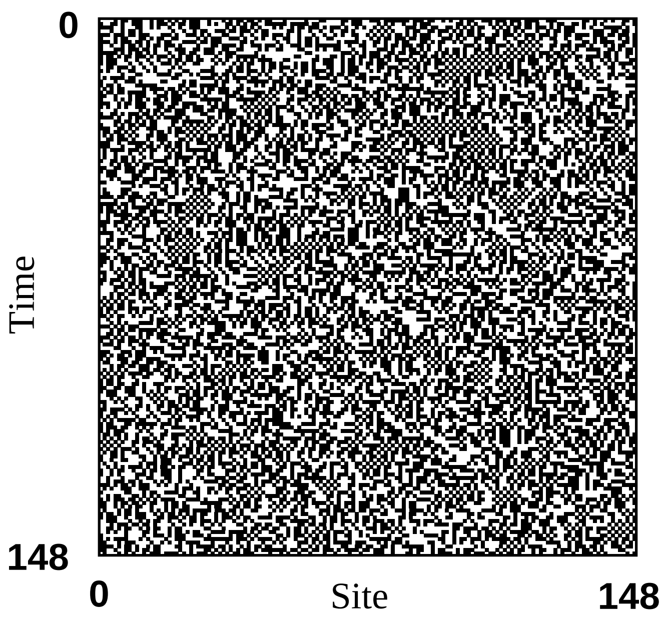

Figure 2: Space-time diagram for a randomly generated $r = 3$ binary-state cellular automaton, iterating on a randomly generated initial configuration. $N = 149$ sites are shown, with time increasing down the page. Here cells with state 0 are white and cells with state 1 are black. (This and the other space-time diagrams given here were generated using the program "la1d" written by James P. Crutchfield.)

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.)

The behavior of one-dimensional binary-state CAs is often illustrated by a "space-time diagram"—a plot of lattice configurations over a range of time steps, with 1s given as black cells and 0s given as white cells and with time increasing down the page. Figure 2 shows such a diagram for a binary-state $r = 3$ CA in which the rule table's output bits were filled in at random. The CA is shown iterating on a randomly generated IC. Apparently structureless configurations, such as those shown in figure 2, are typical for the vast majority of CAs. To produce CAs that can perform sophisticated parallel computations, the GA must search for CAs in which the actions of the cells, taken together, is coordinated so as to produce the desired behavior. This coordination must, of course, happen in the absence of any central processor or central memory directing the coordination.

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).

Design Process

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). If it does, the whole lattice should eventually produce an unchanging configuration of all 1s; otherwise it should eventually go to all 0s. More formally, we call this task the "$\rho_c = \frac{1}{2}$" 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 $\rho_c = \frac{1}{2}$ 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, the task is nontrivial for a small-radius ($r \ll N$) CA, since a small-radius CA relies only on local interactions. It has been argued that no finite-radius, finite-state 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 ($\approx N$). To do this requires the global coordination of cells that are separated by large distances and that cannot communicate directly. How can this be done? Our interest was to see if the GA could devise one or more methods.

The chromosomes evolved by the GA were bit strings representing CA rule tables with $r = 3$. Each chromosome consisted of the output bits of a 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 randomly generated chromosomes (generated with some initial biases—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 either 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 initial density, ρ_0, was never exactly $\frac{1}{2}$, since N was chosen to be odd. 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 of 100 ICs. A different sample was chosen at each generation, making the fitness function stochastic. In addition, 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 distribution across $\rho \in [0, 1]$ (i.e., ICs of each density from $\rho = 0$ to $\rho = 1$ were approximately equally represented). This flat distribution was used because the unbiased distribution is binomially distributed and thus very strongly peaked at $\rho = \frac{1}{2}$. The ICs selected from such a distribution will likely all have $\rho \approx \frac{1}{2}$, the hardest cases to classify. Using an unbiased sample made it very difficult for the GA 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 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. (More details of the implementation are given in Mitchell, Crutchfield, and Hraber 1994.)

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, and section B1.3, this volume). 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 too 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 was a good way of making a "first cut" and allowing rules that survive to be tested over different ICs. Since a new set of ICs was produced every generation, rules that were copied without modification were always retested on this new set. If a rule performed well and thus survived over a large number of generations, then it was likely to be a genuinely better rule than those that were not selected, since it was tested with a large set of ICs.

Results

Three hundred different runs were performed, each starting with a different random-number seed. On most runs the GA evolved a rather unsophisticated class of strategies. One example, a CA here called ϕ_a, is illustrated in figure 3a. 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 seven 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 notion of global coordination or interesting 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 $\rho_c = \frac{1}{2}$ 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. These impediments are discussed in detail in Mitchell, Crutchfield, and Hraber (1994), but the last point merits some elaboration here.

The biased, flat 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, and these ICs did not push the GA hard enough to find better solutions. We are currently exploring a "coevolution" scheme 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 three such rules (each from a different run) are illustrated in figure 3b–d.

For example, ϕ_d was the best-performing rule discovered in our initial GA experiments. In figure 3d it can be seen that, under ϕ_d, there is a transient phase during which 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_d 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 later at a larger scale. 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 program.

The above explanation of how ϕ_d performs the $\rho_c = \frac{1}{2}$ task is an informal one obtained by careful scrutiny of many space-time diagrams. Can we understand more rigorously how the evolved CAs perform the desired computation? Understanding the results of GA evolution is a general problem—typically the GA is asked to find individuals that achieve high fitness but is not told what traits the individuals should have to attain high fitness. One could say that this is analogous to the difficulty biologists have in understanding the products of natural evolution (e.g., us)! We computational evolutionists have similar problems, since we do not specify *what* solution evolution is supposed to create; we ask only that it find *some* good solution. 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 especially difficult in the case of cellular automata, since the emergent computation performed by a given CA is 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 and understanding 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 article. Very briefly, for those familiar with formal language theory, 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. 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 4 hints at this higher level of description: to produce it we filtered the regular domains from the space-time behavior of a GA-evolved CA to leave only the particles and their interactions, in terms of which the emergent algorithm of the CA can be understood.

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 epochs by which highly fit rules were evolved by the GA. An illustration of the succession of these epochs for the synchronization task is given in Figure 5. 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.) The computational mechanics analysis allowed us to understand the evolutionary innovations produced by the GA in the higher-level language of particles and particle interactions as opposed to the low-level language of

CA rule tables and spatial configurations.

Conclusions

The discoveries of rules such as ϕ_b–ϕ_d 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 made by a GA 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. 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, by studying the GA's behavior, we have already learned how evolution's breaking of symmetries can lead to suboptimal computational strategies (Mitchell, Crutchfield, and Hraber 1993); eventually we may be able to use such computer 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 (Crutchfield, 1994), 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 for section G1.15

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).

Forrest, S. 1990. Emergent computation: Self-organizing, collective, and cooperative phenomena in natural and artificial computing networks. *Physica D* 42: 1–11.

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.

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.

Langton, C. G. 1992. Introduction. In C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, eds., *Artificial Life II.* Reading, MA: Addison-Wesley.

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.

Toffoli, T., and Margolus, N. 1987. *Cellular Automata Machines: A New Environment for Modeling.* Cambridge, MA: MIT Press.

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

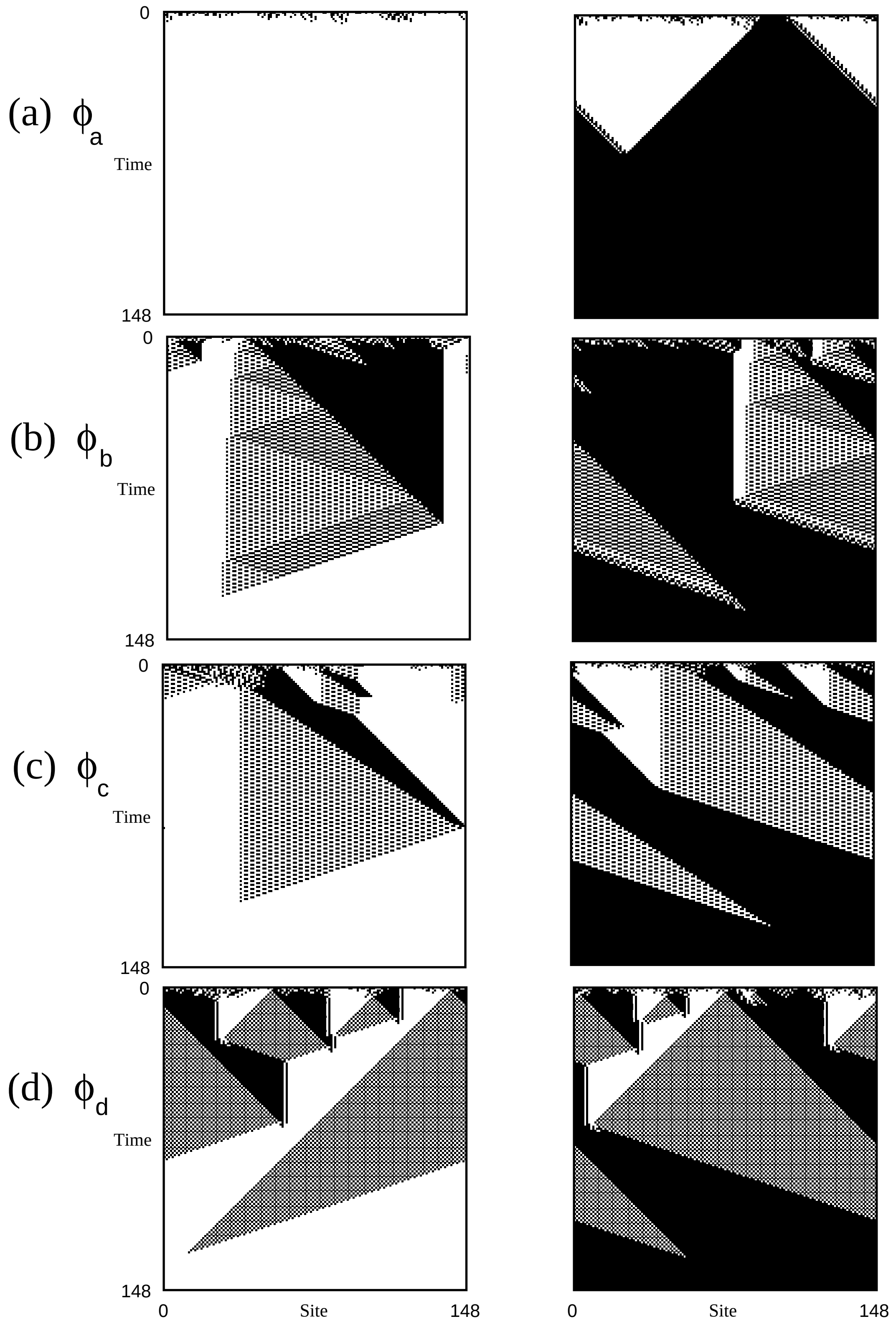

Figure 3: Space-time diagrams from four different rules discovered by the GA (adapted from Das, Mitchell, and Crutchfield 1994). The left diagrams have $\rho_0 < \frac{1}{2}$; the right diagrams have $\rho_0 > \frac{1}{2}$. All are correctly classified. Fitness increases from (a) to (d).

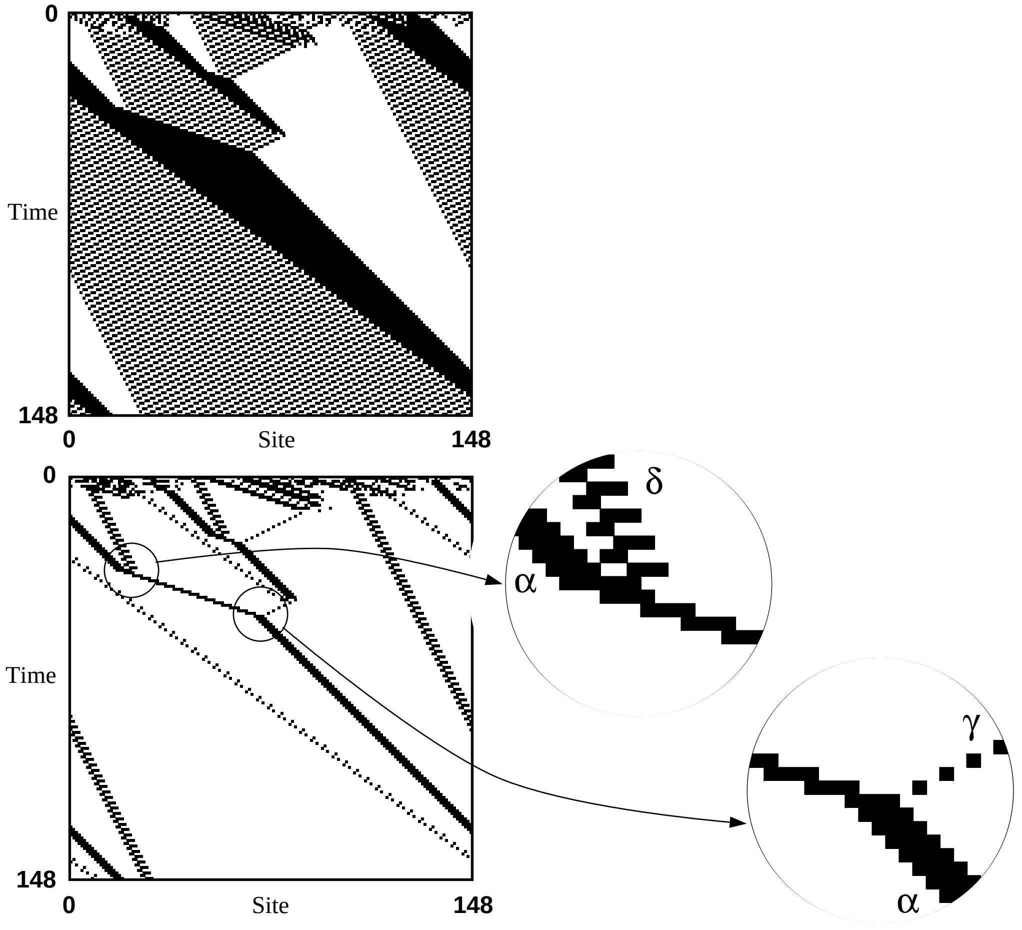

Figure 4: A space-time diagram of a GA-evolved rule for the $\rho_c = \frac{1}{2}$ task, and the same diagram with the regular domains filtered out, leaving only the particles and particle interactions (two of which are magnified). (Reprinted from Crutchfield and Mitchell, 1995).

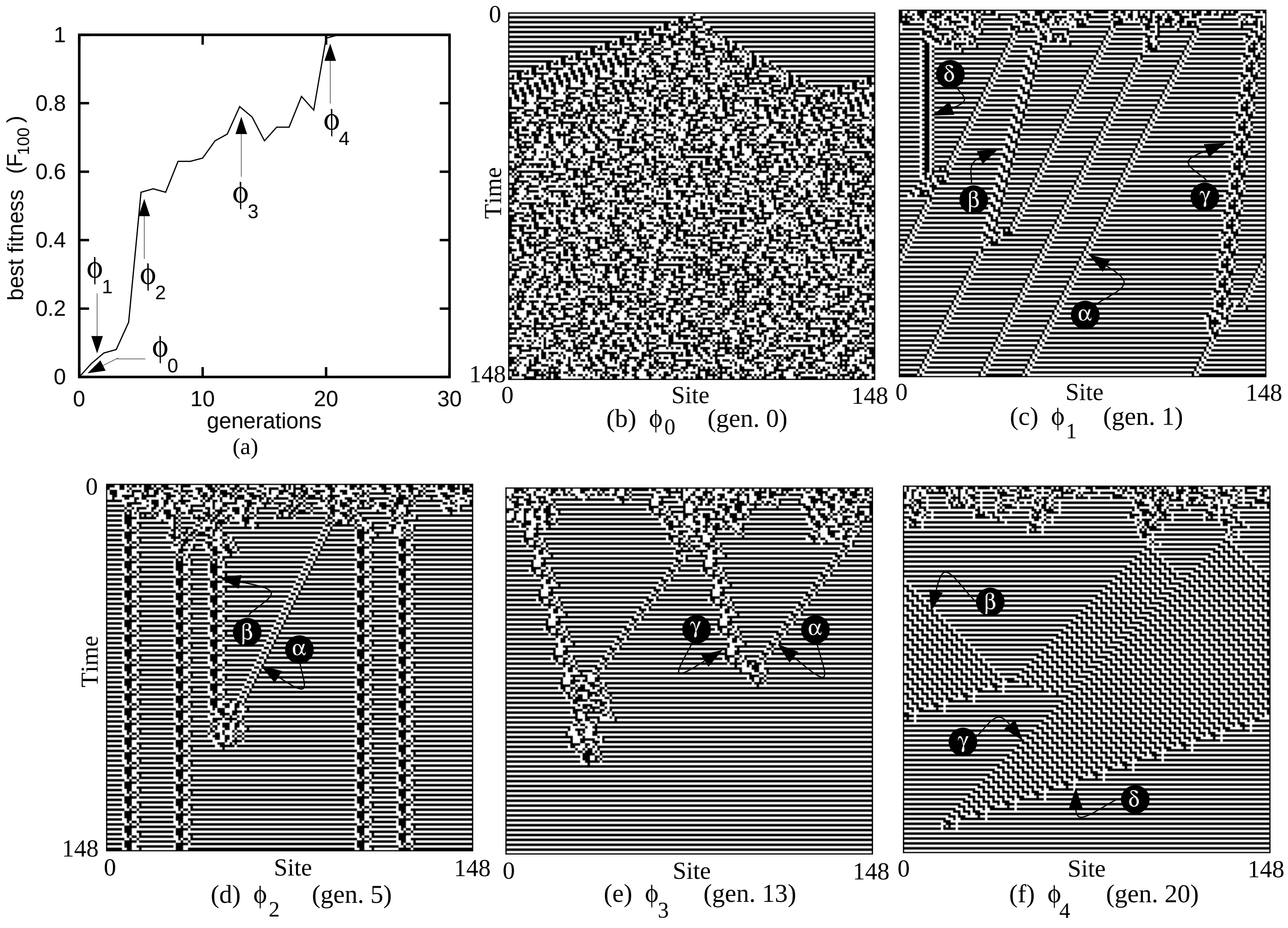

Figure 5: Evolutionary history of one GA run on the synchronization task: (a) F_{100} versus generation for the most fit CA in each population. The arrows indicate the generations in which the GA discovered each new significantly improved strategy. (b)–(f) Space-time diagrams illustrating the behavior of the best ϕ at each of the five generations marked in (a). The ICs are disordered except for (b), which consists of a single 1 in the center of a field of 0s. The same Greek letters in different figures represent different types of particles. (Reprinted from Das, Crutchfield, Mitchell, and Hanson, 1995.)