A Genetic Algorithm Discovers Particle-Based Computation in Cellular Automata

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

Submitted to the *Third Parallel Problem-Solving From Nature Conference*
March 2, 1994

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 [6, 9]). 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 (e.g., individual insects, economic agents, immune-systems cells, or neurons) 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 [2, 3].

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

[1]Santa Fe Institute, 1660 Old Pecos Trail, Suite A, Santa Fe, New Mexico, U.S.A. 87501.
Email: raja@santafe.edu, mm@santafe.edu
[2]Physics Department, University of California, Berkeley, CA, U.S.A. 94720.
Email: chaos@gojira.berkeley.edu

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 [4, 1]. 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 in 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

In words, this rule says that for each neighborhood of three adjacent cells, the new state is decided by a majority vote among the three cells. At time step t, 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 overviews of CA theory and applications, see, e.g., [14, 15]). 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 [10]. 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

$\rho_0 < \rho_c$, then within M time steps the CA should relax to the fixed-point configuration of all 0s; otherwise within M time steps it should relax to the fixed point configuration of all 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 [12], 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 "chromosomes"—bit strings listing the rule-table output bits in lexicographic order of neighborhood patterns. For binary $r = 3$ rules, the chromosomes representing rules are of length $2^{2r+1} = 128$. The size of the rule space the GA searches is thus 2^{128}—far too large for any kind of exhaustive search. The fitness of a rule in the population is calculated by: (i) randomly choosing I ICs that are uniformly distributed over $\rho \in [0.0, 1.0]$, with 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. The rule's fitness $F_I(\phi)$ is the fraction of the I ICs on which the rule 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 [10].

In each generation the GA goes through the following steps. (i) A new set of I ICs is generated. (ii) $F_I(\phi)$ is calculated for each rule ϕ in the population. (iii) The population is ranked in order of fitness. (iv) A number E of the highest fitness ("elite") rules is copied without modification to the next generation. (v) The remaining $P - E$ rules for the next generation are formed by single-point crossovers between randomly chosen pairs of elite rules. The parent rules are chosen from the elite with replacement. The offspring from each crossover are each mutated m times, 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 [13]. (For a discussion of Packard's experiment, see [11].)

The fitness function F_I is a rough estimate of the "exhaustive performance"—the perfor-

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 [5], where the complete look-up tables for these rules are also given.)

mance that would be measured by exhaustively testing a CA on all 2^N ICs. 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 [11]. Allowing M to be larger—up to ten times the lattice size—did not change the qualitative results of the experiments [10]. 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 [11]. 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 difference in the final results was observed). For a more detailed justification of these parameter settings and the results of parameter-modification experiments, see [10, 11].

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 [10, 11], 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 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. The meaning of "sufficiently large" depends on the particular rule implementing one of these strategies, but typically it is close to the neighborhood size.

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. The left space-time diagram has $\rho_0 < 1/2$ and the right one has $\rho_0 > 1/2$. 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 right 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", $\mathcal{P}_I^N(\phi)$, 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 $\rho \approx 0.5$). After each run of the GA we measured $\mathcal{P}_{10^4}^{149}$ for the elite rules in the final generation. The highest measured $\mathcal{P}_{10^4}^{149}(\phi)$ 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 tuned by the GA for $N = 149$. The second row of Table 1 gives $\mathcal{P}_{10^4}^{N}(\phi_{1a})$ 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 $\mathcal{P}_{10^4}^{149} = 0$ for all three values of N (first row of Table 1).

Mitchell, Crutchfield, and Hraber [10] 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 [10] 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 $\mathcal{P}_{10^4}^{N}$ values for these three rules are given in Table 1. As can be seen, $\mathcal{P}_{10^4}^{149}$ 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