Evolving Robust Asynchronous Cellular Automata for the Density Task

Marco Tomassini
Mattias Venzi
Computer Science Institute,
University of Lausanne,
1015 Lausanne, Switzerland

In this paper the evolution of three kinds of asynchronous cellular automata are studied for the density task. Results are compared with those obtained for synchronous automata and the influence of various asynchronous update policies on the computational strategy is described. How synchronous and asynchronous cellular automata behave is investigated when the update policy is gradually changed, showing that asynchronous cellular automata are more adaptable. The behavior of synchronous and asynchronous evolved automata are studied under the presence of random noise of two kinds and it is shown that asynchronous cellular automata implicitly offer superior fault tolerance.

1. Introduction

Cellular automata (CA) [1] are discrete dynamical systems that have been studied theoretically for years due to their architectural simplicity and the wide spectrum of behaviors they present. They have also been used as discrete simulations of physical, chemical, social, and biological systems that are difficult or impossible to model using differential equations or other standard mathematical methods. In this work we concentrate on the commonly assumed hypothesis of simultaneous updating of the CA cells, that is, their *synchronicity*. This update mode is interesting because of its conceptual simplicity and because it is easier to deal with in mathematical terms, which explains why most formal studies of CA have been done for it (e.g., [2, 3] among many others). However, perfect synchronicity is only an abstraction: if CA are to model physical or biological situations or are to be considered physically embodied computing machines then the synchronicity assumption is untenable. In fact, in any spatially extended system signals cannot travel faster than light. Hence, for given dimensions it is impossible for a signal emitted by a global clock to reach any two computing elements at exactly the same time, which poses the problem of latching the signal in order for the units to work in synchronous mode. In biological and sociological environments agents act at different, and possibly uncor-

related, times which seems to preclude a faithful globally synchronous simulation in most cases of interest [4].

In this study we relax the synchronicity constraint and work with various kinds of *asynchronous* CA on a well-known computational problem: density classification. There have been few works on asynchronous CA compared with the synchronous or parallel ones. These studies have tended to show that asynchronous updates often give rise to completely different time evolutions for the CA. For instance, cyclic attractors are no longer possible and generally there is a loss of the rich structures commonly found in synchronous CA. The following references, which include random boolean networks (a generalization of CA) and social simulations, can be consulted [4, 5, 6, 7].

Since asynchronous, rather than synchronous evolution seems to be the norm in nature, one can conclude that asynchronous automata deserve to be studied more fully than has been the case until now. One possible advantage of asynchronous systems is that they appear to be more tolerant to faulty behaviors of various kinds than synchronous ones. Taking into account the increasing miniaturization and the future availability of nano and molecular computational systems with an enormous number of parts, this is a potentially important point that we explore in some detail.

The paper is organized as follows. Section 2 summarizes definitions and facts about standard CA and their asynchronous counterparts. Section 3 deals with the artificial evolution of asynchronous CA for the density task and compares their behavior and solution strategies with those of known synchronous CA. Section 4 presents the results of studying the behavior of synchronous CA when the environment becomes gradually asynchronous and, respectively, of asynchronous CA that become progressively more synchronous. Section 5 examines the degree of tolerance of synchronous and asynchronous CA for the density task to noisy updating of two kinds. Finally, section 6 presents our conclusions and hints to further work and open questions.

2. Synchronous and asynchronous cellular automata

CA are dynamical systems in which space and time are discrete. A standard CA consists of an array of cells, each of which can be in one of a finite number of possible states, updated synchronously in discrete time steps, according to a local, identical interaction rule. Here we will only consider boolean automata for which the cellular state $s \in \{0, 1\}$. The state of a cell at the next time step is determined by the current states of a surrounding neighborhood of cells. The regular cellular array (grid) is d-dimensional, where $d = 1, 2, 3$ is used in practice. In this paper we shall concentrate on $d = 1$, that is, one-dimensional grids. The identical rule contained in each cell is essentially a finite state machine,

Rule table:

neighborhood:	111	110	101	100	011	010	001	000
output bit:	1	1	1	0	1	0	0	0

Grid:

$t = 0$	0	1	1	0	1	0	1	1	0	1	1	0	0	1	1
$t = 1$	1	1	1	1	0	1	1	1	1	1	1	0	0	1	1

Figure 1. Illustration of a one-dimensional, two-state CA. The connectivity radius is $r = 1$, meaning that each cell has two neighbors, one to its immediate left and one to its immediate right. Grid size is $N = 15$. The rule table for updating the grid is shown on top. The grid configuration over one time step is shown at the bottom. The grid is viewed as a circle, with the leftmost and rightmost cells each acting as the other's neighbor.

usually specified in the form of a rule table (also known as the transition function), with an entry for every possible neighborhood configuration of states. The cellular neighborhood of a cell consists of itself and the surrounding (adjacent) cells. For one-dimensional CA, a cell is connected to r local neighbors (cells) on either side where r is referred to as the *radius* (thus, each cell has $2r + 1$ neighbors). When considering a finite-sized grid, spatially periodic boundary conditions are frequently applied, resulting in a circular grid for the one-dimensional case (see Figure 1 for an illustration). Nonuniform (also known as inhomogeneous) CA are the same as uniform ones, the only difference being that the cellular rules need not be identical for all cells. In this paper we will deal with uniform CA exclusively.

A common method of examining the behavior of one-dimensional CA is to display a two-dimensional space-time diagram, where the horizontal axis depicts the configuration at a certain time t and the vertical axis depicts successive time steps (e.g., Figure 2 shown later). The term *configuration* refers to an assignment of ones and zeros at a given time step (i.e., a horizontal line in the diagram). In the figures state 0 is in black, while state 1 is in white.

As stated in the introduction, asynchronous CA are physically more reasonable than parallel ones. However, there are many ways for sequentially updating the cells of a given CA (for an excellent discussion of this point, see the paper by Schönfisch and de Roos [8], from which most of the considerations in this section are inspired). The most general and unbiased way is *independent random ordering* of updates in time, which corresponds to a Poisson stochastic process. We use a close approximation to it which consists of randomly choosing the cell to be updated next, with replacement. This corresponds to a binomial distri-

bution for the update probability and, of course, the limiting case for large n is the Poisson distribution (where n is the number of cells in the grid).

For comparison purposes we also employ two other update methods: *fixed random sweep* and *random new sweep* (we employ the same terms as in [8]). In the fixed random sweep update, each cell to be updated is chosen with uniform probability without replacement; this will produce a certain update sequence $(c_1^j, c_2^k, \ldots, c_n^m)$, where c_t^p means that cell number p is updated at time t and $(j, k, \ldots, m)$ is a permutation of the n cells. The same sequence of cell updates is then used for the following update cycles. The random new sweep method is the same except that each new sweep through the array is done by picking a different random permutation of the cells. In all cases we call a *time step* the process of updating n times, which corresponds to updating all n cells in the grid for the methods without replacement (fixed random sweep and random new sweep) and possibly less than n cells in the binomial method, since some cells might be updated more than once.

It should be noted that because our chosen asynchronous updating is nondeterministic, the same CA may reach a different configuration after n time steps on the same initial distribution of states, which is of course not the case for synchronous CA. This is because the trajectory in configuration space that is followed depends on the evaluation order of the cells for asynchronous CA, while there is a single possible sequence of configurations for a synchronous CA for a given initial configuration of states.

3. Evolving asynchronous cellular automata for the density task

In this section we define the density task and describe how asynchronous CA for performing this task can be evolved using genetic algorithms (GAs). We also deal with the features of the evolutionary process and compare the evolved CA strategies with those observed in the synchronous case.

3.1 The density task

The density task is a prototypical computational task for CA that has been much studied due to its simplicity and richness of behavior. For one-dimensional finite CA of size n (with n odd for convenience) it is defined as follows: The CA must relax to a fixed-point pattern of all 1s if the initial configuration of states contains more 1s than 0s and, conversely, it must relax to a fixed-point pattern of all 0s otherwise, after a number of time steps of the order of the grid size. This computation is trivial for a computer having a central control. Indeed, just scanning the array and adding up the number of, say, 1 bits will provide the answer

in $O(n)$ time. However, the density task is nontrivial for a small radius one-dimensional CA since such a CA can only transfer information at finite speed relying on local information exclusively, while density is a global property of the configuration of states [9]. It has been shown that the density task cannot be solved perfectly by a uniform, two-state CA with finite radius [10], although a slightly modified version of the task can be shown to enjoy perfect solution by such an automaton [11]. At any rate, the lack of a perfect solution does not prevent one from searching for imperfect solutions of as good a quality as possible. In general, given a desired global behavior for a CA (e.g., the density task capability), it is extremely difficult to infer the local CA rule that will give rise to the emergence of the computation sought. This is because of the possible nonlinearities and large-scale collective effects that cannot in general be predicted from the sole local CA updating rule, even if it is deterministic. Since exhaustive evaluation of all possible rules is out of the question except for elementary ($d = 1, r = 1$) automata, one possible solution of the problem consists in using evolutionary algorithms, as first proposed by Packard in [12] and further developed by Mitchell *et al.* in [9, 13] for uniform and synchronous CA and by Sipper for nonuniform ones in [14]. The evolution of nonuniform, asynchronous CA for the density task has been studied in [15].

3.2 Artificial evolution of cellular automata

A popular kind of evolutionary algorithm is a genetic algorithm (GA) [16, 17]. A GA maintains a population of μ encoded tentative solutions to a problem that are competitively manipulated by applying some variation operators to find a satisfactory solution. A GA (see the following pseudo-code) proceeds in an iterative manner by generating new populations of individuals from the old ones. Every individual in the population is the encoded (binary, real, etc.) version of a tentative solution. An evaluation function associates a fitness value to every individual indicating its suitability to the problem. The canonical algorithm applies stochastic operators such as selection, crossover, and mutation on an initially random population in order to compute a whole generation of new individuals. The termination condition is usually set to reach a preprogrammed number of iterations of the algorithm, or to find an individual with a given error if the optimum, or an approximation to it, is known beforehand.

```
generation = 0
Seed Population and evaluate individuals
while not termination condition do
    generation = generation + 1
    Evaluation
    Selection
```

Crossover
Mutation
end while

We use a GA similar to the one described in [9] for synchronous CA, with the aim of evolving asynchronous CA for the density task. Each individual in the population represents a candidate rule and is represented simply by the output bits of the rule table in lexicographic order of the neighborhood (see section 2). Here $r = 3$ has been used, which gives a chromosome length of $2^{2r+1} = 128$ and a search space of size 2^{128}, which is far too large to be searched exhaustively. The population size is 100 individuals, each represented by a 128-bit string, initially randomly generated from a uniform density distribution over the interval $[0, 1]$. The fitness of a rule in the population has been calculated by randomly choosing 100 out of the 2^n possible initial configurations (ICs) with uniform density; that is, any configuration has the same probability of being selected, and then iterating the rule on each IC for $M = 2n$ time steps, where $n = 149$ is the grid size. The rule's fitness is the fraction of ICs for which the rule produced the correct fixed point, given the known IC density. At each generation a different set of ICs is generated for each rule. After ranking the rules in the current population according to their fitness, the 20% top rules are copied in the next population without change. The remaining 80 rules are generated by crossover and mutation. Crossover is single-point and is performed between two individuals randomly chosen from the top 20 rules with replacement and is followed by single-bit mutation of the two offspring. The best 80 rules after the application of the genetic operators enter the new population.

The performance of the best rules found at the end of the evolution is evaluated on a larger sample of ICs and it is defined as the fraction of correct classifications over 10^4 randomly chosen ICs. Moreover, the ICs are sampled according to a binomial distribution (i.e., each bit is independently drawn with probability 1/2 of being 0). Clearly, this distribution is strongly peaked around $\rho_0 = 1/2$ and thus it makes a much more difficult case for the CA (ρ_0 is defined to be the density of 0s in the IC).

3.3 Evolutionary dynamics and results: Synchronous cellular automata

Mitchell and coworkers performed a number of studies on the emergence of synchronous CA strategies for the density task during evolution [9, 13]. Their results are significant since they represent one of the few instances where the dynamics of emergent computation in complex, spatially extended systems can be understood. In summary, these findings can be subdivided into those pertaining to the evolutionary history and those that are part of the "final" evolved automata. For the former, they essentially observed that, in successful evolution

experiments, the fitness of the best rules increases in time according to rapid jumps, giving rise to what they call "epochs" in the evolutionary process. Each epoch corresponds roughly to a new, increasingly sophisticated solution strategy. Concerning the final CA produced by evolution, it was noted that, in most runs, the GA found unsophisticated strategies that consisted in expanding sufficiently large blocks of adjacent 1s or 0s. This "block-expanding" strategy is unsophisticated in that it mainly uses local information to reach a conclusion. As a consequence, only those IC that have low or high density are classified correctly since they are more likely to have extended blocks of 1s or 0s. In fact, these CA have a performance of around 0.6. However, some of the runs gave solutions that presented novel, more sophisticated features that yielded better performance (around 0.77) on a wide distribution of ICs. These new strategies rely on traveling signals that transfer spatial and temporal information about the density in local regions through the lattice. An example of such a strategy is given in Figure 2, where the behavior of the so-called GKL rule is depicted [9]. The GKL rule is hand-coded but its behavior is similar to that of the best solutions found by evolution. In spite of the relative success of the GA, there exist hand-coded CA that have better performance, such as the GKL rule. On the other hand, Andre *et al.* in [18] have been able to artificially evolve a CA that is as good as the best manually-designed CA by using genetic programming. Although they used a great deal more computational resources than Mitchell and coworkers,

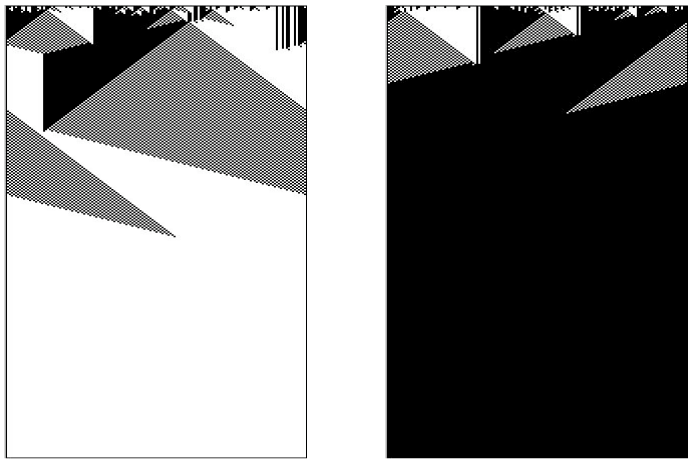


Figure 2. Space-time diagram for the GKL rule. The density of zeros ρ_0 is 0.476 in (a) and $\rho_0 = 0.536$ in (b). State 0 is depicted in black; 1 in white.

this nevertheless shows that artificial evolution is a viable solution for this problem.

Crutchfield and coworkers have developed sophisticated methodologies for studying the transfer of long-range signals and the emergence of computation in evolved CA. This framework is known as "computational mechanics" and it describes the intrinsic CA computation in terms of regular domains, particles, and particle interactions. Details can be found in [19, 20].

3.4 Evolutionary dynamics and results: Asynchronous cellular automata

For the evolution of asynchronous CA we have used GA parameters as described in section 3.2. Due to the high computational cost, we have performed 15 runs, each lasting for 100 generations, for each of the asynchronous update policies. This is not enough to reach very good results, but it is sufficient for studying the emergence of well-defined computational strategies, which has been our main objective here. As expected, the evolved asynchronous CA find it more difficult to solve the density task due to their stochastic nature. In fact, a given CA could classify the same IC in a different way depending on the update sequence, and indeed, although synchronous CA are delocalized systems, because of the presence of a global clock, a kind of central control is still present, which is not the case for asynchronous CA. Nevertheless, for all the asynchronous update methods, CA with fair capabilities were evolved. In Table 1 we list the best rules found by the GA for the three update modes. We note that the performance of the solutions are lower than the corresponding figures for synchronous CA.

The behavior of the CA evolved with all three asynchronous updating modes were very similar both from the point of view of performance, as well as from the point of view of the solution strategies that evolved. Since independent random ordering, that is, uniform update, is in some sense the more natural, for reasons of space we will mainly describe it here, although most of what we say also applies to the other two methods.

During most evolutionary runs in all asynchronous methods we observed the presence of periods in the evolution in which the fitness of the

Update Mode	Rule	Performance
Uniform	00024501006115AF5FFFBFFDE9EFF95F	67.2
Fixed Random	1140040602024141505 77E771F55FFFF	67.7
Random New	00520140006013264B7DFCDF4F6DC7DF	65.5

Table 1. Performance of the best evolved asynchronous rules calculated over 10^4 binomially distributed ICs. Rule numbers are in hexadecimal.

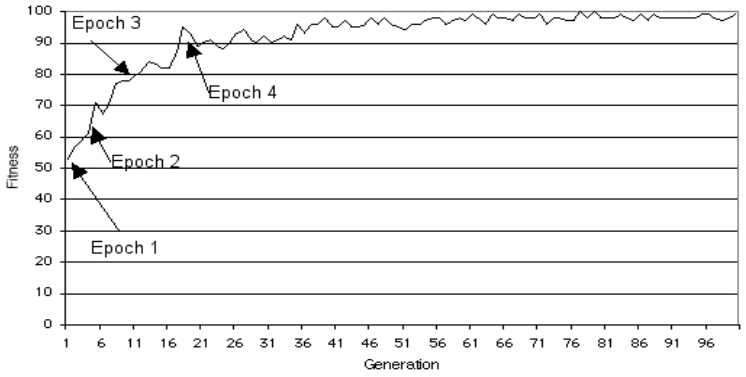


Figure 3. Epochs of innovation in the evolution of uniform choice asynchronous CA for the density task.

best rules increase in rapid jumps (see Figure 3). These "epochs" were observed in the synchronous case too (see section 3.3) and correspond to distinct computational innovations, that is, to major changes in the strategies that the CA uses for solving the task.

In epoch 1 the evolution discovers local naive strategies that only work on "extreme" densities (i.e., low or high) but most often not on both at the same time, as one can see in Figure 4.

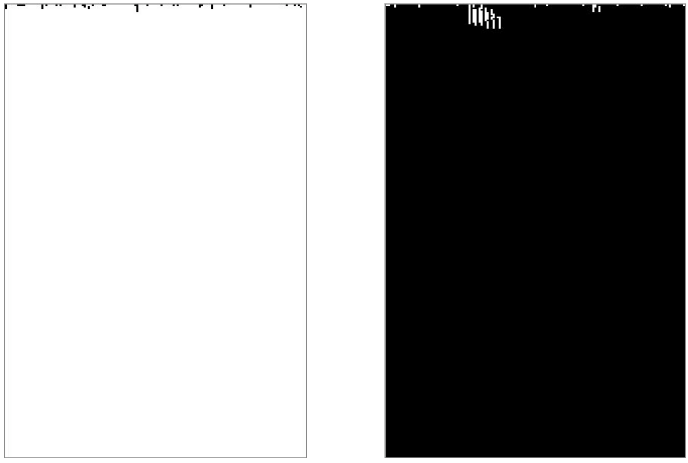
In the following epoch 2, rules specialize on low or high densities as well and use unsophisticated strategies, but now they give correct results on both low and high densities. This can be seen, for instance, in Figure 5.

In epoch 3, with fitness values comprised between 0.80 and 0.90, one sees the emergence of block-expanding strategies, as in the synchronous case, but more disordered here. Moreover, narrow vertical strips make their appearance (see Figure 6). These strips are the main mechanism by which fully evolved successful CA are able to classify many intermediate cases and approximately solve the density problem.

The following, and last, epoch 4 sees the refinement of the vertical strips strategy with fitness above 0.9 and steadily increasing. The propagating patterns become less noisy and the strategy is little affected by the intrinsic stochasticity of the update rule. Figure 7 illustrates the best solution found by evolution at the end of epoch 4. The "zebra-like" moving patterns, which represent the most efficient strategies for evolved asynchronous automata, are different from those found in the synchronous case. In fact, the asynchronous updating modes have the effect of destroying or delaying the propagation of the long-range transversal signals that carry information in the synchronous case (see Figure 2). Thus, the CA expands 1^* and 0^* blocks, which collide and

(a) (b)

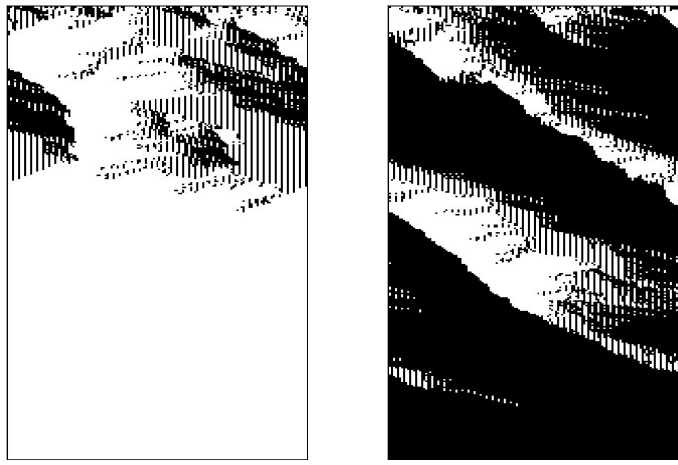
Figure 4. Space-time diagrams for an epoch 1 rule. (a) $\rho_0 = 0.107$, (b) $\rho_0 = 0.912$. Clearly, the classification in (b) is incorrect.



(a) (b)

Figure 5. Space-time diagrams for an epoch 2 rule. (a) $\rho_0 = 0.194$, (b) $\rho_0 = 0.879$. The rule only classifies low or high densities.

annhilate. As a result, small blocks propagate in time, which gives the characteristic zebra-like patterns. These strips are stable and propagate further to the right or to the left.



(a) (b)

Figure 6. Space-time diagrams for an epoch 3 rule. (a) $\rho_0 = 0.489$, (b) $\rho_0 = 0.510$. Block-expanding and vertical strips make their appearance.

(a) (b)

Figure 7. Space-time diagrams for the best asynchronous rule found. The density $\rho_0 = 0.55$ in both (a) and (b) and the initial state configuration is the same. The different time evolution is due to the nondeterminism of the updating policy.

4. Merging the synchronous and asynchronous worlds

We have seen that evolved synchronous CA for the density task have a rather better performance than asynchronous ones, as it was expected if one takes into account their deterministic nature. Now, although paral-

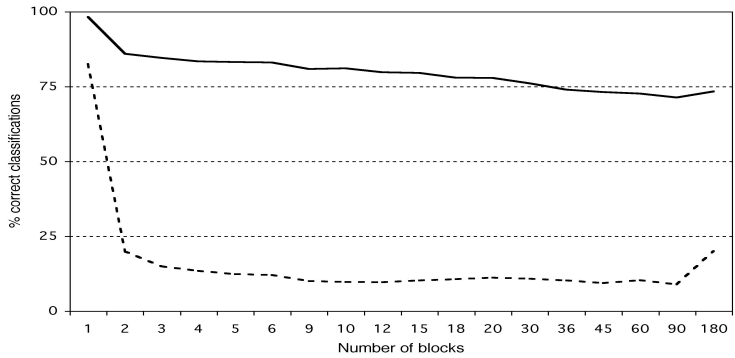


Figure 8. Percent of correct classifications as a function of the number of blocks in the grid for two choices of ICs for the GKL rule. Dashed line: binomial distribution; thick line: uniform distribution over density in $[0, 1]$.

lel update is infeasible, one could obtain a more realistic approximation by subdividing the whole grid into blocks of $c \leq n$ cells each that are updated synchronously within the block, while the blocks themselves are updated asynchronously. Thus, if the number of blocks varies from 1 to n the system will go from complete synchrony to complete asynchrony ($n = 180$ here). Let us start with synchronous CA rules becoming progressively asynchronous (random new sweep is used for whole block updating). Both the best evolved rule as well as the GKL rule gave very poor results. They are extremely sensitive to perturbations since even a small amount of noise destroys the strict synchronization carried by the propagating transversal signals (see section 3.3). This can be seen in Figure 8, where the performance of the GKL rule is shown against the number of blocks for two distributions of ICs. Performance remains acceptable for ICs chosen uniformly between 0 and 1 but it totally degrades for a binomial distribution, which is the more difficult and interesting case.

Figure 9 depicts the space-time diagram of the GKL rule with two and 10 asynchronous blocks respectively. One sees clearly that, with two blocks already, signals are prevented from traveling and do not combine at block boundaries, as would be the case if all the cells were updated in parallel (see Figure 2 for comparison), which explains why the CA is incapable of performing the task.

Starting now from the other end of the spectrum, we consider the best asynchronous CA rule going progressively more synchronous (i.e., from right to left in Figure 10). In this case we see that performances are progressively lower but the loss is gradual, and only for the extreme cases of a few large blocks does performance approach 0.5.

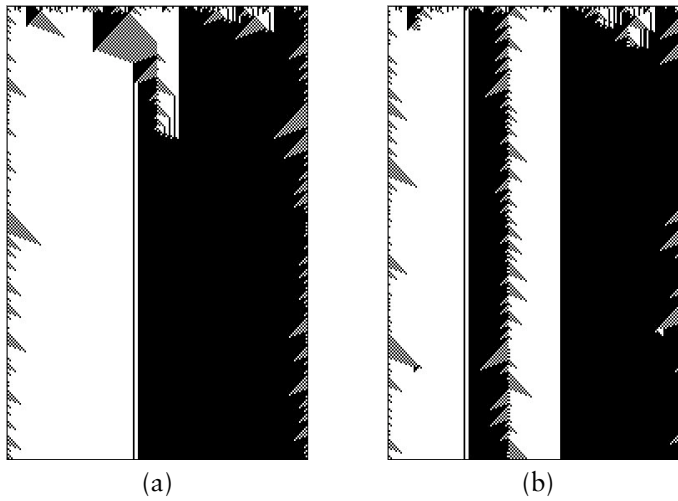


Figure 9. Asynchronous behavior of the GKL rule with two (a) and 10 (b) blocks respectively.

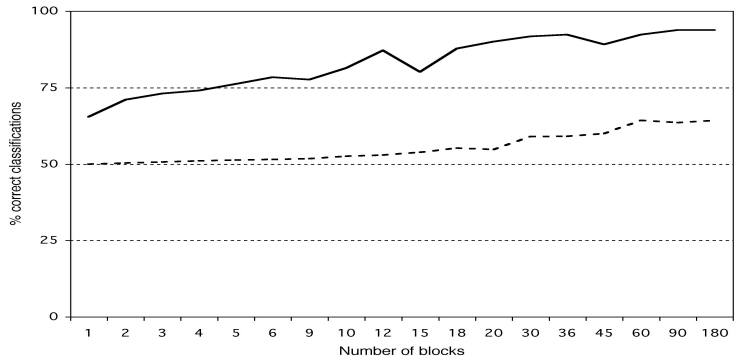


Figure 10. Percent of correct classifications as a function of the number of blocks in the grid for two choices of ICs for the best asynchronous rule. Dashed line: binomial distribution; thick line: uniform distribution over density in $[0, 1]$.

Figure 11 depicts the case of 90 and 60 blocks and shows that the strategy of solution is not perturbed in these cases.

We can thus conclude that, although synchronous and asynchronous rules have been evolved, respectively, in synchronous and asynchronous environments, asynchronous rules adapt better to changes; in other words, they are more robust.

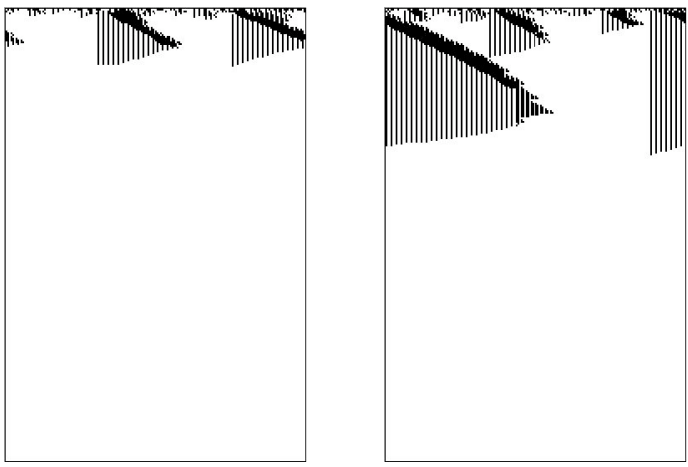


Figure 11. Synchronous behavior of the best asynchronous rule with (a) 90 and (b) 60 blocks respectively. The density $\rho_0 = 0.444$ in both (a) and (b).

5. The effect of noise

There are at least two possible sources of indeterminism in CA: one is random asynchronous updating, as explained in the previous sections, the second is faulty functioning of the rules, or of the cells, or both. In this section we explore in some detail the behavior of evolved CA in the face of noise of the second type. These kinds of considerations will be important in the future, when it is likely that self-organizing computational systems composed of an enormous number of parts will be used. Fault tolerance was an important issue in the beginning of the computer era because of the high unreliability of the then used computing elements. Future components will be more reliable but, because there will be very large numbers of them, even if the individual probability of failure is low, the overall probability of having a fault at any given time will be high. Thus, it will be important to design systems that are partially or totally tolerant to such faults. Of course, the comparatively small and simple systems studied here are only toys with respect to real future computing machines. Nonetheless, their study is certainly a worthy first step. It should be noted that we will not try to correct or compensate for the errors, which is an important but very complicated issue. Rather, we will focus on the self-recovering capabilities of the systems under study.

We will study two kinds of perturbations and the way in which they affect the density task, the first is *probabilistic updating* and the second is *intermittent faults*. They are defined as follows.

- *Probabilistic updating.* A CA rule may yield the incorrect output bit with probability p_f, and thus the probability of correct functioning will be $(1 - p_f)$. Futhermore, we assume that errors are uncorrelated.
- *Intermittent faults.* At time t a given cell has a certain probability of being inactive; that is, of keeping its current state. Cells may fail independently of each other.

For probabilistic updating, usually two initially identical copies of the system are maintained. One evolves undisturbed with $p_f = 0$, while the second is submitted to a nonzero probability of fault. One can then measure such things as Hamming distances between unperturbed and faulty configurations, which give information on the spreading of damage (e.g., [21] and references therein, where the case of synchronous, nonuniform CA is examined). Rather than presenting Hamming distances, which are a global measure and thus do not yield much insight into the local processes that cause performance loss, we prefer to show the results of typical evolutions. Figure 12 depicts the typical behavior of the best evolved synchronous rule under noise, this rule is called EvCA here [9]. We see that only for extremely low ($p_f = 0.00001$) values of the fault probability does the CA correctly classify density. For higher values, either the classification is incorrect (c) or the CA is so perturbed that it cannot accomplish the task any longer (d). Clearly, the propagation of transversal signals is more and more hindered as the automaton becomes more noisy.

Figure 13 shows the same evolution for the best evolved asynchronous CA using the uniform choice update policy. Visual inspection already indicates that the CA is much less perturbed by random noise in the rules. Even relatively high levels of faults do not prevent the CA from recovering and finding the correct classification in many cases. This is clearly due to the fact that asynchronous CA were evolved in a noisy environment (the randomness associated with the sequential update order) and thus, to some extent, this allows them to cope better with errors.

Although the previous examples are single cases, they are typical of what happens. Figure 14 shows a histogram of the ratio of the success rate of the best evolved synchronous CA (EvCA [9]) and of two evolved asynchronous CA as a function of the fault probability, with respect to the unperturbed versions. Each CA has been tested on 1000 ICs. The ranking is relative, since we only kept the successful runs of the unperturbed automata to calculate the ratio. One sees clearly that, already for $p_f = 1.0 \times 10^{-4}$, the synchronous CA starts to degrade, while both asynchronous versions keep good performance, especially the one with uniform choice, up to p_f values of the order of 10^{-3}.

In the case of intermittent fault we have tested 1000 ICs for each of a number of probability values of cell inactivity. We have used three CA rules: the best evolved synchronous CA (EvCA [9]), the GKL rule,

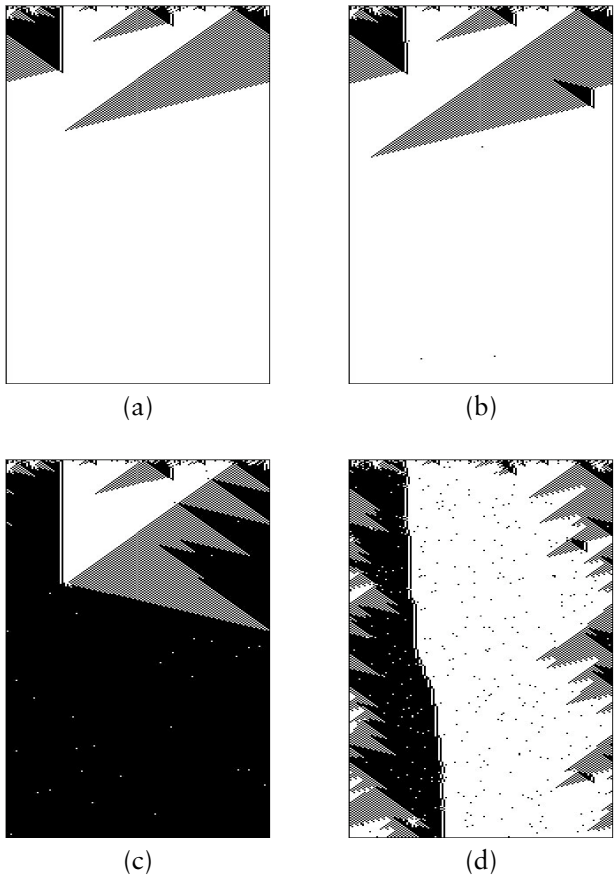


Figure 12. Typical behavior of EvCA [9] under probabilistic updating. The density ρ_0 is 0.416 and the probabilities of fault p_f in (a), (b), (c), and (d) are, respectively, 0, 0.0001, 0.001, and 0.01.

and the best evolved uniform choice asynchronous automaton. The results are reported in Figure 15. We observe that for low values of the fault probability the three rules are almost equivalent in that they keep a very good level of performance. However, as soon as the probability exceeds 0.01, the two synchronous rules collapse, especially GKL, while the asynchronous rule does not seem to suffer much from the increasing level of noise and keeps a good performance level in the whole range, except for high probability values (note the logarithmic scale on the horizontal axis).

Once again, the results of this section confirm that asynchronous CA degrade much more gracefully than synchronous ones in noisy environments and thus they intrinsically offer more resilience and robustness.

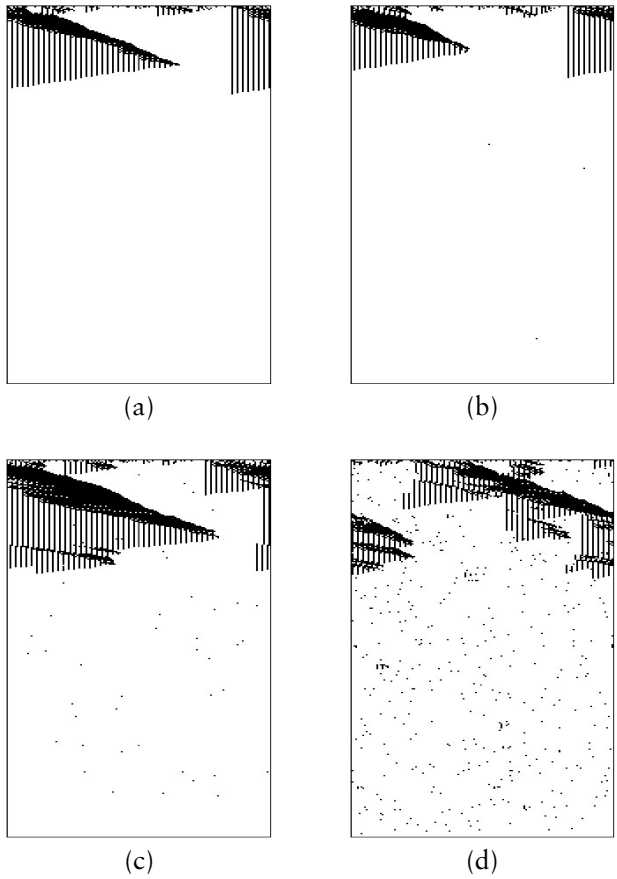


Figure 13. Typical behavior of an asynchronous CA under probabilistic updating. The density ρ_0 is 0.416 and the probabilities of fault p_f in (a), (b), (c), and (d) are, respectively, 0, 0.0001, 0.001, and 0.01.

It would be interesting to evolve CA with some noise added, to see whether their fault-tolerant capabilities are enhanced.

6. Conclusions

In this work we have shown that physically more realistic asynchronous cellular automata (CA) of various kinds can be effectively evolved for the density task using genetic algorithms (GAs), although their performance is lower than that obtained by evolved synchronous CA. We have also shown that the computational strategies discovered by the GA in the asynchronous case are different from those of synchronous CA due to the presence of a stochastic component in the update. This very reason

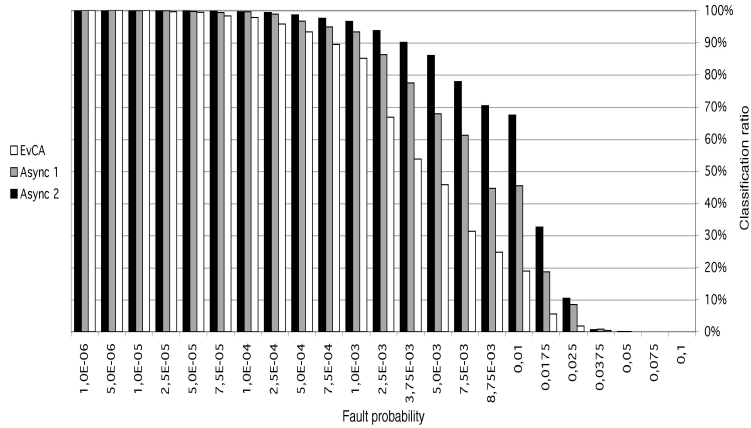


Figure 14. Histogram representing the percentage of success of three noisy automata with respect to the unperturbed versions. Only the perfect runs have been retained for the unperturbed automata. The probability of fault is on the horizontal axis. Async 1 (grey bar) is the new random sweep automaton, while Async 2 (black bar) corresponds to the uniform choice CA. EvCA (white bar) is the best evolved synchronous CA.

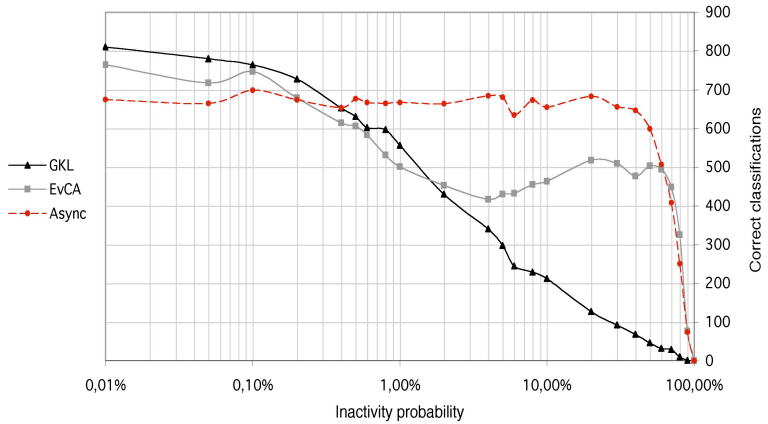


Figure 15. Number of correct classifications as a function of inactivity probability. The curves refer to the GKL rule and to two asynchronous CA (see text).

makes them more resistant to changes in the environment and thus potentially more interesting as computational devices in the presence of noise. Other important aspects that we are studying, but are not included here, are the scalability properties of evolved CA and further investigations into their fault-tolerance aspects.

Acknowledgment

We acknowledge the Fonds National Suisse pour la recherche scientifique for financial support under the grant 21-58893.99.

References

[1] T. Toffoli and N. Margolus, *Cellular Automata Machines* (The MIT Press, Cambridge, MA, 1987).

[2] H. Gutowitz, editor, *Cellular Automata: Theory and Experiment* (The MIT Press, Cambridge, MA, 1991).

[3] S. Wolfram, *Cellular Automata and Complexity* (Addison-Wesley, Reading, MA, 1994).

[4] B. A. Huberman and N. S. Glance, "Evolutionary Games and Computer Simulations," *Proceedings of the National Academy of Sciences USA*, **90** (1993) 7716–7718.

[5] H. Bersini and V. Detour, "Asynchrony Induces Stability in Cellular Automata Based Models," in *Artificial Life IV*, edited by R. A. Brooks and P. Maes (The MIT Press, Cambridge, MA, 1996).

[6] I. Harvey and T. Bossomaier, "Time Out of Joint: Attractors in Asynchronous Random Boolean Networks," in *Proceedings of the Fourth European Conference on Artificial Life*, edited by P. Husbands and I. Harvey (The MIT Press, Cambridge, MA, 1997).

[7] T. E. Ingerson and R. L. Buvel, "Structure in Asynchronous Cellular Automata," *Physica D*, **10** (1984) 59–68.

[8] B. Schönfisch and A. de Roos, "Synchronous and Asynchronous Updating in Cellular Automata," *BioSystems*, **51** (1999) 123–143.

[9] M. Mitchell, P. T. Hraber, and J. P. Crutchfield, "Revisiting the Edge of Chaos: Evolving Cellular Automata to Perform Computations," *Complex Systems*, **7** (1993) 89–130.

[10] M. Land and R. K. Belew, "No Perfect Two-state Cellular Automata for Density Classification Exists," *Physical Review Letters*, **74**(25) (1995) 5148–5150.

[11] M. S. Capcarrere, M. Sipper, and M. Tomassini, "Two-state, $r = 1$ Cellular Automaton that Classifies Density," *Physical Review Letters*, **77**(24) (1996) 4969–4971.

[12] N. H. Packard, "Adaptation Toward the Edge of Chaos," in *Dynamic Patterns in Complex Systems*, edited by J. A. S. Kelso, A. J. Mandell, and M. F. Shlesinger (World Scientific, Singapore, 1988).

[13] M. Mitchell, J. P. Crutchfield, and P. T. Hraber, "Evolving Cellular Automata to Perform Computations: Mechanisms and Impediments," *Physica D*, **75** (1994) 361–391.

[14] M. Sipper, *Evolution of Parallel Cellular Machines: The Cellular Programming Approach* (Springer-Verlag, Heidelberg, 1997).

[15] M. Sipper, M. Tomassini, and M. S. Capcarrere, "Evolving Asynchronous and Scalable Non-uniform Cellular Automata," in *Proceedings of International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA97)*, edited by G. D. Smith, N. C. Steele, and R. F. Albrecht (Springer-Verlag, Vienna, 1997).

[16] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, third edition (Springer-Verlag, Heidelberg, 1996).

[17] M. Mitchell, *An Introduction to Genetic Algorithms* (The MIT Press, Cambridge, MA, 1996).

[18] D. Andre, F. H. Bennett III, and J. R. Koza, "Discovery by Genetic Programming of a Cellular Automata Rule that is Better than any Known Rule for the Majority Classification Problem," in *Genetic Programming 1996: Proceedings of the First Annual Conference*, edited by J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo (The MIT Press, Cambridge, MA, 1996).

[19] J. E. Hanson and J. P. Crutchfield, "Computational Mechanics of Cellular Automata: An Example," *Technical Report 95-10-95* (Santa Fe Institute Working Paper, 1995).

[20] W. Hordijk, J. P. Crutchfield, and M. Mitchell, "Mechanisms of Emergent Computation in Cellular Automata," in *Parallel Problem Solving from Nature: PPSN V*, volume 1498 of *Lecture Notes in Computer Science*, edited by A. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel (Springer-Verlag, Heidelberg, 1998).

[21] M. Sipper, M. Tomassini, and O. Beuret, "Studying Probabilistic Faults in Evolved Non-uniform Cellular Automata," *International Journal of Modern Physics C*, 7(6) (1996) 923–939.