International Journal of Modern Physics C, Vol. 7, No. 6 (1996) 923–939

STUDYING PROBABILISTIC FAULTS IN EVOLVED NON-UNIFORM CELLULAR AUTOMATA

MOSHE SIPPER
Logic Systems Laboratory, Swiss Federal Institute of Technology
IN-Ecublens, CH-1015 Lausanne, Switzerland
E-mail: Moshe.Sipper@di.epfl.ch

MARCO TOMASSINI
Logic Systems Laboratory, Swiss Federal Institute of Technology, and
Institute of Computer Science, University of Lausanne
E-mail: Marco.Tomassini@di.epfl.ch

OLIVIER BEURET
Logic Systems Laboratory, Swiss Federal Institute of Technology
IN-Ecublens, CH-1015 Lausanne, Switzerland
E-mail: Olivier.Beuret@di.epfl.ch

Received 31 October 1996
Revised 6 November 1996

We study the effects of random faults on the behavior of one-dimensional, non-uniform cellular automata (CA), where the local update rule need not be identical for all grid sites. The CA systems examined were obtained via an approach known as *cellular programming*, which involves the evolution of non-uniform CAs to perform non-trivial computational tasks. Using the "system replicas" methodology, involving a comparison between a perfect, non-perturbed version of the CA and a faulty one, we find that our evolved systems exhibit graceful degradation in performance, able to tolerate a certain level of faults. We then "zoom" into the fault-tolerant zone, where "good" computational behavior is exhibited, introducing measures to fine-tune our understanding of the faulty CAs' operation. We study the error level as a function of time and space, as well as the recuperation time needed to recover from faults. Our investigation reveals an intricate interplay between temporal and spatial factors, with the presence of different rules in the grid giving rise to complex dynamics. Studies along this line may have applications to future computing systems that will contain thousands or even millions of computing elements, rendering crucial the issue of resilience.

Keywords: Non-Uniform Cellular Automata; Cellular Programming; Fault Tolerance; Damage Spreading.

1. Introduction

Cellular automata (CA) are discrete dynamical systems containing a finite or infinite number of simple components that interact locally.[1,2] Each component can be considered a lattice site in a low-dimensional grid space having only a finite number

of possible states. In synchronous, uniform CAs the values of all the sites in the grid are updated simultaneously at each discrete time step according to a given identical rule that depends on the state of the site itself and on that of a small number of neighboring lattice points. Non-uniform CAs can also be considered in which the local update rule need not be identical for all grid sites.[3–5]

Though simple in their definition and elementary components, some CAs have been shown to be capable of complex global behavior, even in one dimension, including chaotic phenomena and universal computation.[6,7] CAs have been widely used in the past to model natural and social phenomena and as computing machines in domains where global behavior arises from local interactions, e.g., for low-level image processing.[8,9]

CAs are massively parallel systems amenable to hardware implementation due to the simplicity of basic components (cells) as well as the local cellular connectivity. Most classical software and hardware systems, especially parallel ones, exhibit a very low level of fault-tolerance, i.e., they are not resilient in the face of errors; indeed, where software is concerned, even a single error can often bring an entire program to a grinding halt. Future computing systems may contain thousands or even millions of computing elements (e.g., Ref. 10). For such large numbers of components, the issue of resilience can no longer be ignored since faults will be likely to occur with high probability. Networks of automata exhibit in principle some fault-tolerance. As an example one can cite artificial neural networks, many of which show graceful degradation in performance when presented with noisy input; furthermore, the malfunction of a neuron or damage to a synaptic weight causes but a small change in the system's overall behavior, rather than bringing it to a complete standstill. Cellular computing systems, such as CAs, may thus be seen as a simple and convenient framework within which to study the effects of such errors.

In this paper we study the effects of random faults on the behavior of one-dimensional CAs that perform given computational tasks. The CA systems examined were obtained via an approach known as *cellular programming*, which involves the evolution of non-uniform CAs to perform non-trivial computational tasks.[11–16] In particular, we are interested in the systems' behavior as a function of the error level; we wish to learn whether there exist error-rate regions in which the automata can be considered to perform their task in an "acceptable" manner. Moreover, the amount and speed of *recovery* after the appearance of a fault is quantified and measured. We also observe how disturbances spread throughout the system to learn under what conditions the perturbation remains limited and does not propagate to the entire system.

In the next section related fault studies in cellular systems are briefly reviewed, followed by a section describing our CA systems and the computational measures employed. We then present the results obtained, ending with concluding remarks.

2. Previous Work on Faults and Damage in Lattice Models

The question of how errors spread and propagate in cooperative systems has been studied in a variety of fields. Given the difficulty of creating analytical models for but the simplest systems, most investigations have been conducted by computer simulation, especially in the area of statistical physics of many-body systems. One system that has received much attention is Kauffman's model, which consists of a non-uniform CA with irregular connectivity in which each cell (lattice site) follows a transition rule that is a random boolean function of the state of its neighbors; the rules as well as the connections between cells are randomly selected at the outset and then remain fixed throughout the system's run.[17] The system has been observed to converge toward limit cycles; it can be perturbed by "mutations," which are random changes of rules. Stauffer[18] and other researchers have studied the spreading of damage on various kinds of two-dimensional lattices as a function of the probability p of mutating rules within the grid. Critical values of p have been found at which a phase transition seems to occur; above the critical p the damage spreads to the entire lattice, while below it the system is stable with respect to damage spreading.

Another well-known system in which the time evolution of damage has been investigated is the Ising ferromagnet and related spin systems. In these "thermal systems" transition probabilities are a function of the temperature. Reference 19 employed Monte Carlo simulations using Metropolis dynamics, finding that there exists a critical temperature T_c, above which (i.e., at high temperatures) an initial damage at a few sites spreads rapidly to the entire system, while below T_c the damage eventually dissipates. Some apparent inconsistencies in this work, due to the use of different transition probability functions, have been resolved in Ref. 20.

The general objective of the kind of research summarized above is the study of the temporal limit behavior of the system as a function of some parameter, such as the probability of fault or the thermal noise. For some systems critical behavior has been shown to occur and in some cases critical exponents were computationally determined. A recent review of damage dynamics in collective systems from the point of view of computational physics can be found in Ref. 21.

3. Computational Tasks and Probabilistic Faults in Cellular Automata

Although the simulation methodology is similar, the main difference between the studies described in the previous section and the work presented herein stems from the fact that we focus on CAs that perform a specified *computational task*, rather than on the long-term dynamics of a physical system under given constraints. From our computational point of view, what is important is the way in which the *task performance* is affected when the system is perturbed.

Programming a CA to execute a given task or to simulate a certain physical system is in general a difficult endeavor. This results from the local dynamics of the system, which renders the design of local interaction rules to perform global computational tasks extremely arduous. Normally the correspondence, or

approximate correspondence, between CA rules and the desired global dynamics has to be found by ingenuity or trial and error. Recently, an alternative approach has been suggested, which consists of applying a process of artificial evolution to "search" for the CA rules necessary to implement a prespecified task.[11–16,22–24]

The details of these methods can be found in the cited references, the general idea being as follows: one starts with a *population* of arbitrary, randomly-assigned rules that are evaluated according to the quality, or *fitness*, of the corresponding CA on the task at hand. Rules that perform better are *selected* and *recombined*, random *mutations* being occasionally applied to maintain population diversity. This evolutionary process may eventually converge toward rules that are "good enough," if not optimal (for recent general reviews on artificial evolution, the reader is referred to Refs. 25–27). The advantage of this methodology is that little design work is needed since the evolutionary process automatically finds suitable rules. To date, several CAs have been evolved to perform diverse computational tasks, including random number generation[14,15] and image processing.[28,29] Two different algorithms have been used to evolve CAs. The standard genetic algorithm used by Ref. 23 gives rise to uniform CAs, whereas the algorithm of Ref. 11, known as *cellular programming*, involves non-uniform ones. The latter was found to produce *quasi-uniform* systems, meaning that only a few distinct rules exist in the grid upon termination of the evolutionary process.

We next introduce the CAs that are the subject of our study. We concentrate on one-dimensional, non-uniform CAs with two possible states per cell (denoted 0, 1), and connectivity radius $r = 1$, meaning that each cell is connected only to its immediate left and right neighbors. Spatially periodic boundary conditions are used, resulting in a circular grid. We have applied the cellular programming evolutionary algorithm to evolve such CAs to perform a number of computational tasks, two of which shall be considered herein, density and synchronization. The one-dimensional density task is to decide whether or not the initial configuration[a] contains more than 50% 1s, relaxing to a fixed-point pattern of all 1s if the initial density of 1s exceeds 0.5, and all 0s otherwise. In the one-dimensional synchronization task the CA, given any initial configuration, must reach a final configuration, within M time steps, that oscillates between all 0s and all 1s on successive time steps. It should be emphasized that both tasks comprise non-trivial computational problems for a small radius CA ($r \ll N$, where N is the grid size), since a global configuration is to be attained in a locally-connected structure, thereby necessitating some form of global information propagation (e.g., via emergent computation).[23,24] Note that the density task cannot be perfectly solved by a uniform, two-state CA, as proven in Ref. 30.[b]

[a] The term "configuration" refers to an assignment of states to cells in the grid.

[b] This result applies to the above statement of the problem, where the CA's final pattern (i.e., output) is specified as a fixed-point configuration. Interestingly, it has recently been proven that by changing the output specification, namely the final pattern toward which the system should converge, a two-state, $r = 1$ uniform CA exists that can perfectly solve the density problem.[31]

We used cellular programming to evolve non-uniform CAs to perform these tasks, attaining high performance for the density task, and perfect performance for the synchronization task.[c] The operation of non-uniform CAs that were evolved to perform these tasks is shown in Fig. 1.[d]

The above CAs evolve[e] in time according to prescribed (evolved) *deterministic* rules; however, noise can be introduced into CA rules, thereby rendering them non-deterministic. For example, for a two-state CA, at each time step the value that is the output of a given deterministic rule can be reversed with probability p_f, denoted the *fault probability*, each site being treated independently of the others (Fig. 1). Thus, a cell updates its state in a non-deterministic manner, setting it at the next time step to that specified in the rule table, with probability $1 - p_f$, or the complementary state, with probability p_f. This definition of noise will be used in what follows since it reasonably models the functioning of a multi-component machine in which the computing elements are subject to stochastic transient faults. Other kinds of perturbations are possible, such as sites becoming unavailable ("permanent damage") or switching to another rule for a long, possibly indefinite, period of time. It is also possible to consider the flipping of site states, either single sites or clusters of sites. Moreover, each site may be updated at each time step according to one rule with probability p and according to a second rule with probability $1-p$.[3] The perturbed Kauffman automata,[18] in which a site selects its rule probabilistically, to be then subjected to random mutations, is an example similar to ours.

The simulation methodology is based on the concept of "system replicas."[21,32,33] Two systems run in parallel, the original unperturbed one ($p_f = 0$), and a second system subjected to a non-zero probability of error ($p_f > 0$). Both systems start with the same initial configuration at time $t = 0$, after which their temporal behavior is monitored and the Hamming distance between the two configurations at each time step is recorded.[f] This provides us with insight into the faulty CA's behavior, by measuring the amount by which it diverges from a "perfect" computation. Our studies are stochastic in nature, involving a number of measures which are obtained experimentally, averaged over a large number of initial configurations.

[c]The different performance measures used, as well as the precise results obtained, are delineated, e.g., in Refs. 11 and 12. Essentially, performance concerns the percentage of input configurations, over a large random sample, for which a correct response is attained, as well as the number of time steps until convergence to the correct final pattern. The "perfect performance" attained for the synchronization task is meant in a stochastic sense since we cannot exhaustively test all 2^N possible initial configurations nor are we in possession to date of a formal proof; nonetheless, we have tested our best-performance CAs on numerous configurations, for all of which synchronization was attained.

[d]The CAs discussed in this paper are fully specified in the Appendix.

[e]Note that we use the term 'evolve' in two distinct ways, the first referring to the artificial evolution of CA rules, while the second refers to a CA's evolution in time. This is done in order to conform with existing terminology; the appropriate meaning can be determined from the context.

[f]The Hamming distance between two configurations is the number of bits by which they differ.

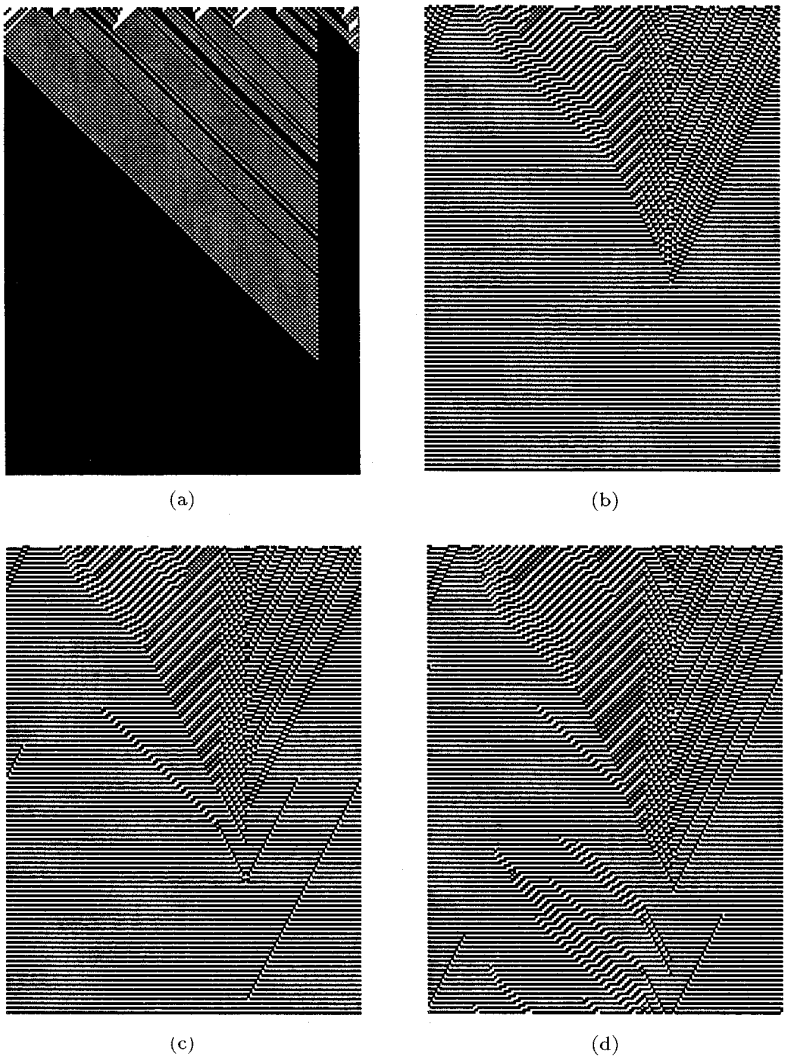


Fig. 1. The operation of evolved, non-uniform, $r = 1$ CAs. Grid size is $N = 149$. White squares represent cells in state 0, black squares represent cells in state 1. The pattern of configurations is shown for the first 200 time steps, with time increasing down the page. The initial configurations were generated by randomly setting the state of each grid cell to 0 or 1 with uniform probability. (a) A CA that was evolved to perform the density task. The operation of a "perfect" system is shown, i.e., with fault probability $p_f = 0$. Initial density of 1s > 0.5. (b) A CA that was evolved to perform the synchronization task. $p_f = 0$. (c) The CA of (b) is run with $p_f = 0.0001$. (d) The CA of (b) is run with $p_f = 0.001$.

The non-uniform CAs studied are ones that have evolved via cellular programming to perform either the density or synchronization tasks, with our fault-tolerance investigation picking up upon *termination* of the evolutionary process. Figures 1(c) and 1(d) depict the operation of an evolved, non-uniform CA on the synchronization task for two different non-zero p_f values.

4. Results

Figure 2 depicts the average Hamming distance as a function of the fault probability p_f. We note that the curve is sigmoid-shaped, exhibiting three observable regions: a slow-rising slope ($p_f \leq 0.0005$), followed by a sharp one ($0.0005 < p_f \leq 0.01$), ending with an attenuated slope ($p_f > 0.01$); this latter region exhibits an extremely large Hamming distance, signifying an unacceptable level of computational error. The most important result concerns the first (left-hand) region, which can be considered the *fault-tolerant zone*, where the faulty CA operates in a near-perfect manner. This demonstrates that our evolved CAs exhibit "graceful degradation" in the face of errors. We also note that there is no essential difference between the two tasks, density and synchronization, except for the higher error level in the "unacceptable" zone attained by the density CAs. These simulations (as well as the rest reported in this section) were repeated several times, obtaining virtually identical results.

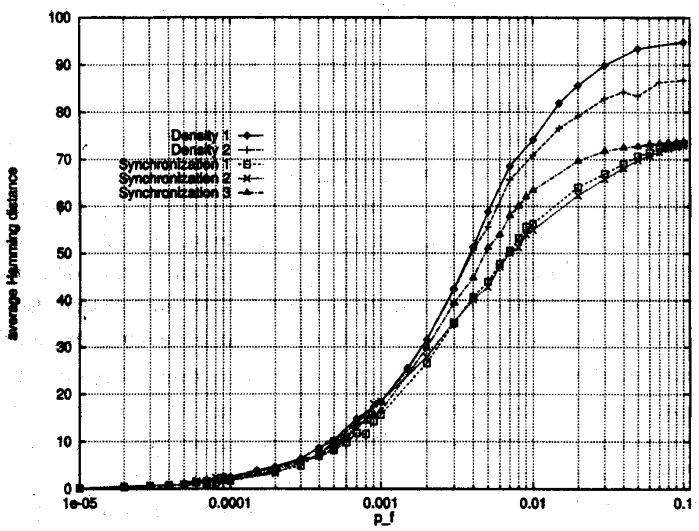


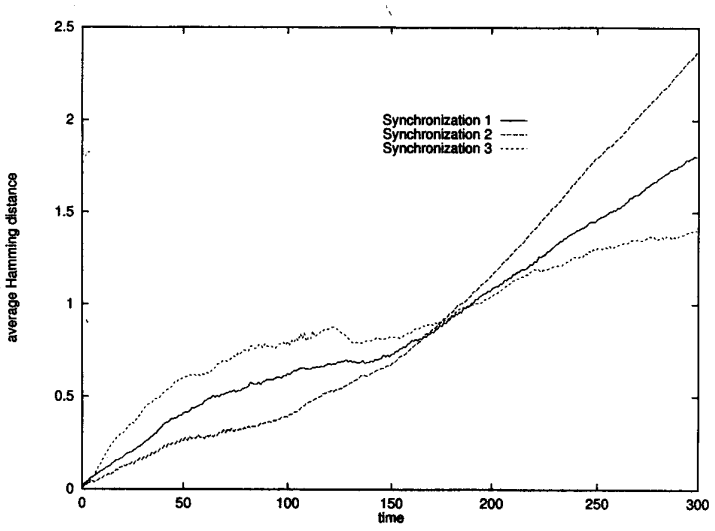
Fig. 2. Average Hamming distance versus fault probability p_f. Five CAs were studied — two that were evolved to perform the density task, and three that were evolved to perform the synchronization task. Grid size is $N = 149$. For each p_f value the CA under test was run on 1000 randomly generated initial configurations for 300 time steps. At each time step the Hamming distance between the "perfect" CA and the faulty one is recorded. The average over all configurations and all time steps is represented as a point in the graph.

The above measure furnishes us with our first glimpse into the workings of the faulty CAs, demonstrating an important global characteristic, namely their ability to tolerate a certain level of faults. We now wish to "zoom" into the fault-tolerant zone, where "good" computational behavior is exhibited, introducing measures to fine-tune our understanding of the faulty CAs' operation. In what follows we shall concentrate on one task, synchronization, due to the improved evolved performance results in comparison to the density task, obtained for the deterministic versions of the CAs (see previous section).[g] We now wish to study the propagation of errors in time; toward this end we examine the Hamming distance between the perfect and faulty versions, as a function of time (step). Our results are depicted in Fig. 3. We note that while Hamming distance is limited within the region suggested by Fig. 2, there are differences between the CAs. Most notable is the high error rate attained by CA 2 in the last 100 time steps.

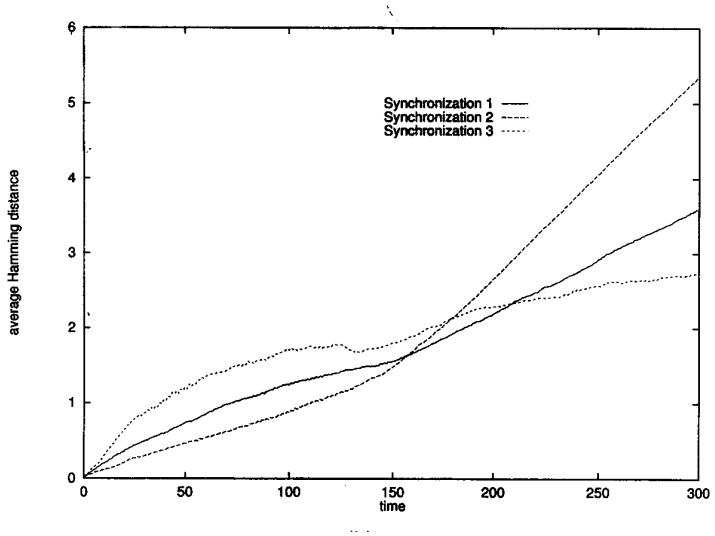
Further investigation revealed that this is due to *critical zones*. These are specific rules or rule blocks (i.e., blocks of cells containing the same rule) that cause an "avalanche" of error spreading, which may eventually encompass the entire system, as demonstrated in Fig. 4. Figure 4(a) shows that the CA's error rate peaks around cell 60, which is at the border of rule blocks (see Appendix). Indeed, when this cell is perturbed (Fig. 4(b)), the error may eventually spread to the entire system, resulting in the diminished performance in later time steps, evident in Fig. 3. Interestingly, this CA has the lowest error rate for the initial part of the computation (Fig. 3). CA 3 exhibits the opposite time behavior, starting with a higher error rate, which increases, however, more slowly (Fig. 3). Figure 5(a) shows that this CA exhibits an error peak at the proximity of cell 90, a much sharper one than that of CA 2, resulting in error containment. Again, cell 90 is at the border of two rule blocks (see Appendix). Figure 5(a) exhibits a minimum at cell 16, which is also a border cell (between rule blocks), demonstrating that such border rules may act in the opposite manner, "stifling" error spreading rather than enhancing it. CA 1 consists of two major rule blocks, exhibiting different error dispersion behavior, as demonstrated in Fig. 6. Thus, by introducing time and space measures, we have shown that although all three CAs are within the fault-tolerant zone, their behavior is quite different.

The final issue we consider is that of *recuperation time*. Since our CAs are in effect computational systems, we wish to learn not only whether they recover from faults but also how long this takes. Toward this end we introduced the following measure: the CA of size $N = 149$ is run for 600 time steps with a given fault probability p_f. If the Hamming distance between the perfect and faulty versions passes a certain threshold, which we have set to $0.05N$ bits, at time t_1, and then falls below this threshold at time t_2, staying below for at least three time steps, then recuperation time is defined as $t_2 - t_1$. Note that such "windows" of faulty

[g]Note that applying the performance measures mentioned in the previous section to the deterministic versions of the three evolved synchronization CAs discussed herein revealed no differences between them.



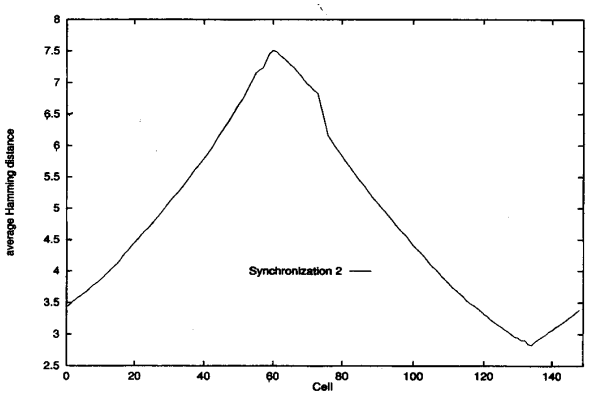
(a)



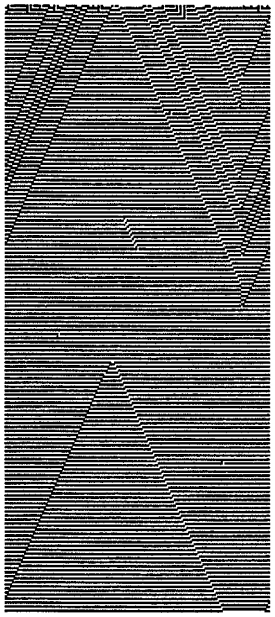
(b)

Fig. 3. Hamming distance as a function of time for three CAs that were evolved to perform the synchronization task. Grid size is $N = 149$. Each CA is run on 1000 random initial configurations for 300 time steps; the Hamming distance per time step is averaged over these configurations. (a) $p_f = 0.00005$. (b) $p_f = 0.0001$.

behavior may occur more than once during the CA's run (i.e., during the 600 time steps); also note that t_2 may equal 600 if the CA never recovers. Simply put, this measure indicates the proportional amount of time that the CA is within a window of unacceptable error level. Our results are depicted in Fig. 7. For $p_f < 0.0001$

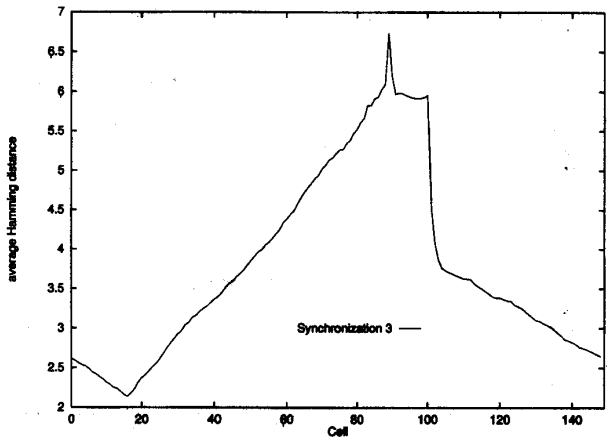


(a)

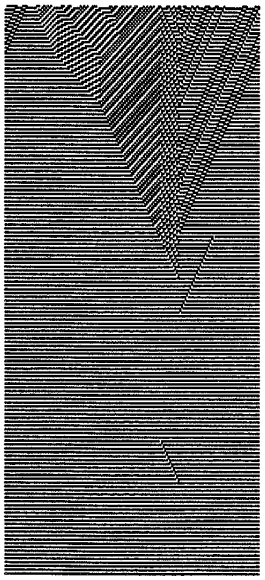


(b)

Fig. 4. Synchronization CA 2: Critical zones. (a) Hamming distance per cell (averaged over 1000 random initial configurations, each run for 300 time steps). Note the peak around cell 60 (the leftmost cell is numbered 0). (b) Perturbing this cell causes an "avalanche" of error spreading. The figure depicts the operation of the CA upon presentation of a random initial configuration; after approximately 200 time steps cell 60's state is flipped. This cell is situated at the border of rule blocks (see Appendix). $p_f = 0.0001$ for both (a) and (b). Grid size is $N = 149$.

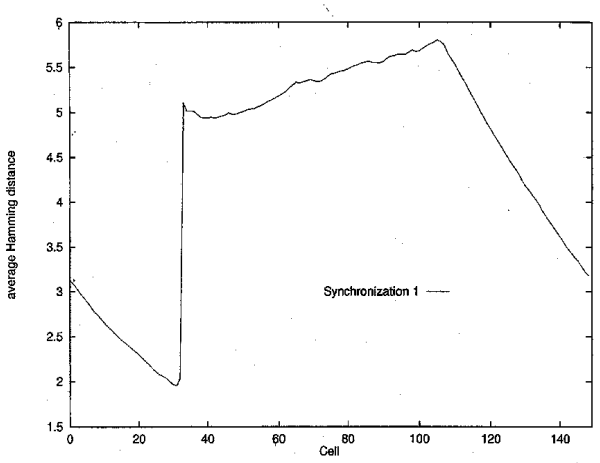


(a)

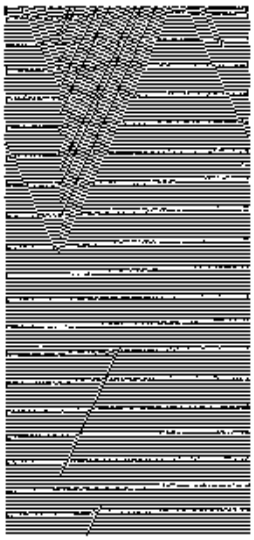


(b)

Fig. 5. Synchronization CA 3. (a) Hamming distance per cell (averaged over 1000 random initial configurations, each run for 300 time steps). Note the peak around cell 90, much sharper than that of Fig. 4. (b) Perturbing this cell does not cause an "avalanche" and the error remains contained. This results in a lower Hamming distance as function of time (Fig. 3). The figure depicts the operation of the CA upon presentation of a random initial configuration; after approximately 200 time steps cell 90's state is flipped. This cell is situated at the border of rule blocks (see Appendix). $p_f = 0.0001$ for both (a) and (b). Grid size is $N = 149$.



(a)



(b)

Fig. 6. Synchronization CA 1. (a) Hamming distance per cell (averaged over 1000 random initial configurations, each run for 300 time steps). Two major rule blocks are present, each exhibiting a different error dispersion behavior, the highest error level being that of the "middle" block (note that the left and right blocks contain the same rule, as can be seen in the Appendix, and therefore constitute one block due to the grid's circularity). (b) Three cells are perturbed, in different parts of the grid (cells 20, 70, 120). The error introduced in the middle block propagates, whereas the other two are immediately stifled. The figure depicts the operation of the CA upon presentation of a random initial configuration; after approximately 200 time steps the states of the above three cells are flipped. $p_f = 0.0001$ for both (a) and (b). Grid size is $N = 149$.

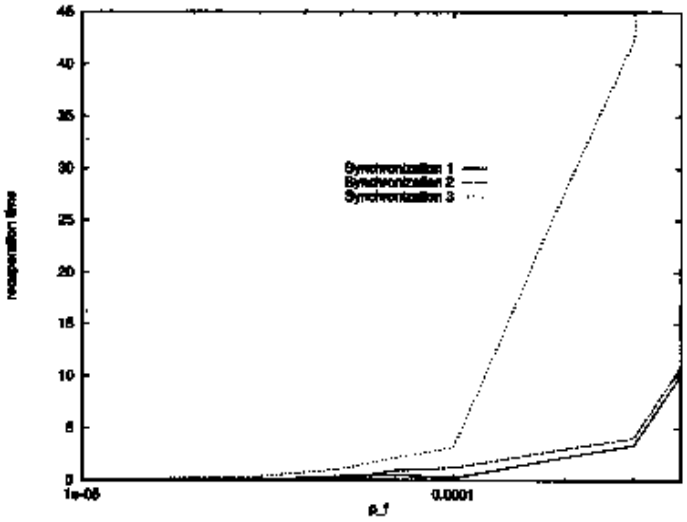


Fig. 7. Recuperation time as a function of fault probability p_f. Each of the three evolved CAs was run on 1000 random initial configurations for 600 time steps. Average results are depicted in the graph. Grid size is $N = 149$.

recuperation time is quite short for all three CAs, however, above this fault level, CA 3 exhibits notably higher recuperation time than the other two. It is interesting in that this CA has the lowest error level over time (Fig. 3).[h] Thus, it is more robust to errors in general, however, certain faults may cause severe problems in terms of recuperation time. This result, along with the others obtained above, demonstrates the intricate interplay between temporal and spatial factors in our evolved non-uniform CAs.

5. Concluding Remarks

We studied the effects of random faults on the behavior of one-dimensional, non-uniform CAs that perform given computational tasks. The CA systems examined were obtained by an artificial evolution approach, known as cellular programming. Using the "system replicas" methodology, involving a comparison between a perfect, non-perturbed version of the CA and a faulty one, we found that our evolved systems exhibit graceful degradation in performance, able to tolerate a certain level of faults. We then zoomed into the fault-tolerant zone, where "good" computational behavior is exhibited, introducing measures to fine-tune our understanding of the faulty CAs' operation. We studied the error level as a function of time and space, as well as the recuperation time needed to recover from faults.

[h] Though Fig. 3 shows results for $p_f \leq 0.0001$, we have verified that the same qualitative behavior is exhibited for $p_f > 0.0001$.

Our study of evolved non-uniform CAs performing computational tasks revealed an intricate interplay between temporal and spatial factors, with the presence of different rules in the grid giving rise to complex dynamics. Clearly we have only taken the first step, and there is much yet to be explored. Other types of measures can be considered, such as fault behavior as a function of grid size, permanent faults along with their effects with respect to the rules distribution within the grid, and "total damage time," i.e., the time required for all sites to be damaged at least once. Another interesting issue involves the introduction of faults during the evolutionary process itself to see how well evolution copes with such non-deterministic CAs. Future computing systems may contain thousands or even millions of computing elements; for such large numbers of components, the issue of resilience can no longer be ignored since faults will be likely to occur with high probability.

Evolving cellular systems hold potential both scientifically, as vehicles for studying phenomena of interest in the domain of complex adaptive systems, as well as practically, showing a range of potential future applications ensuing the construction of adaptive systems. We hope this paper has shed some light on the behavior of such systems under faulty conditions.

Appendix. Specification of the Evolved Non-Uniform CAs

This appendix specifies the five non-uniform, $r = 1$ CAs discussed in the paper, evolved via cellular programming. The listing includes the rule found in each cell, where rule numbers are given in accordance with Wolfram's convention,[33] representing the decimal equivalent of the binary number encoding the rule table. All grid sizes are $N = 149$. Cell 0 is the leftmost cell.

Synch. 1:

From cell	To cell	Rule
0	32	31
33	105	83
106	106	19
107	148	31

Density 1:

From cell	To cell	Rule
0	39	226
40	40	234
41	71	226
72	72	234
73	142	226
143	144	224
145	148	226

Synch. 2:

From cell	To cell	Rule
0	55	21
56	56	85
57	58	21
59	60	53
61	73	63
74	132	31
133	148	21

Density 2:

From cell	To cell	Rule
0	106	226
107	108	224
109	131	226
132	132	234
133	148	226

Synch. 3:

From cell	To cell	Rule
0	15	53
16	16	55
17	29	59
30	89	43
90	100	39
101	101	7
102	148	53

References

1. S. Wolfram, "Universality and complexity in cellular automata," *Physica* **D10**, 1 (1984).
2. T. Toffoli and N. Margolus, *Cellular Automata Machines* (The MIT Press, Cambridge, Massachusetts, 1987).
3. G. Y. Vichniac, P. Tamayo, and H. Hartman, "Annealed and quenched inhomogeneous cellular automata," *Journal of Statistical Physics* **45**, 875 (1986).
4. H. Hartman and G. Y. Vichniac, "Inhomogeneous cellular automata," in *Disordered Systems and Biological Organization*, eds. E. Bienenstock, F. Fogelman, and G. Weisbuch (Springer-Verlag, Berlin, 1986), pp. 53–57.
5. M. Sipper, "Non-uniform cellular automata: Evolution in rule space and formation of complex structures," in *Artificial Life IV*, eds. R. A. Brooks and P. Maes, (The MIT Press, Cambridge, Massachusetts, 1994), pp. 394–399.
6. S. Wolfram, *Cellular Automata and Complexity* (Addison-Wesley, 1994).
7. K. Lindgren and M. G. Nordahl, "Universal computation in simple one-dimensional cellular automata," *Complex Systems* **4**, 299 (1990).
8. K. Preston, Jr. and M. J. B. Duff, *Modern Cellular Automata: Theory and Applications* (Plenum Press, New York, 1984).

9. A. Broggi, V. D'Andrea, and G. Destri, "Cellular automata as a computational model for low-level vision," *International Journal of Modern Physics* **C4**, 5 (1993).
10. K. E. Drexler, *Nanosystems: Molecular Machinery, Manufacturing and Computation* (John Wiley, New York, 1992).
11. M. Sipper, "Co-evolving non-uniform cellular automata to perform computations," *Physica* **D92**, 193 (1996).
12. M. Sipper and E. Ruppin, "Co-evolving architectures for cellular machines," *Physica* **D**, 1996 (to appear).
13. M. Sipper and E. Ruppin, "Co-evolving cellular architectures by cellular programming," in *Proceedings of IEEE Third International Conference on Evolutionary Computation, ICEC'96* (1996), pp. 306–311.
14. M. Sipper and M. Tomassini, "Co-evolving parallel random number generators," in H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel (editors), *Parallel Problem Solving from Nature — PPSN IV*, volume 1141 of *Lecture Notes in Computer Science* (Springer-Verlag, Heidelberg, 1996), pp. 950–959.
15. M. Sipper and M. Tomassini, "Generating parallel random number generators by cellular programming," *International Journal of Modern Physics* **C7**, 181 (1996).
16. M. Sipper, "Evolving uniform and non-uniform cellular automata networks," to appear in *Annual Reviews of Computational Physics, Volume V*, ed. D. Stauffer (World Scientific, 1997).
17. S. A. Kauffman, *The Origins of Order* (Oxford University Press, New York, 1993).
18. D. Stauffer, "Computer simulations of cellular automata," *Journal of Physics A: Mathematical and General* **24**, 909 (1991).
19. H. E. Stanley, D. Stauffer, J. Kertész, and H. J. Herrmann, "Dynamics of spreading phenomena in two-dimensional Ising models," *Physical Review Letters* **59**, 2326 (1987).
20. A. Coniglio, L. de Arcangelis, H. J. Herrmann, and N. Jan, "Exact relations between damage spreading and thermodynamical properties," *Europhysics Letters* **8**, 315 (1989).
21. N. Jan and L. de Arcangelis, "Computational aspects of damage spreading," in *Annual Reviews of Computational Physics, Volume I*, ed. D. Stauffer (World Scientific, 1994), pp. 1–16.
22. J. P. Crutchfield and M. Mitchell, "The evolution of emergent computation," in *Proceedings of the National Academy of Sciences USA* **92**, 10742 (1995).
23. M. Mitchell, J. P. Crutchfield, and P. T. Hraber, "Evolving cellular automata to perform computations: Mechanisms and impediments," *Physica* **D75**, 361 (1994).
24. R. Das, J. P. Crutchfield, M. Mitchell, and J. E. Hanson, "Evolving globally synchronized cellular automata," in *Proceedings of the Sixth International Conference on Genetic Algorithms*, ed. L. J. Eshelman (Morgan Kaufmann, San Francisco, CA, 1995), pp. 336–343.
25. M. Tomassini, "A survey of genetic algorithms," in *Annual Reviews of Computational Physics, Volume III*, ed. D. Stauffer (World Scientific, 1995), pp. 87–118. Also available as: Technical Report 95/137, Department of Computer Science, Swiss Federal Institute of Technology, Lausanne, Switzerland, July, 1995.
26. M. Tomassini, "Evolutionary algorithms," in *Towards Evolvable Hardware*, Volume 1062 of *Lecture Notes in Computer Science*, eds. E. Sanchez and M. Tomassini (Springer-Verlag, Berlin, 1996), pp. 19–47.
27. Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd ed. (Springer-Verlag, Berlin, 1996).
28. M. Sipper, "Designing evolware by cellular programming," to appear in *Proceedings of The First International Conference on Evolvable Systems: from Biology to*

Hardware (ICES96), Lecture Notes in Computer Science (Springer-Verlag, Heidelberg, 1996).
29. M. Sipper, "The evolution of parallel cellular machines: Toward evolware," *BioSystems*, 1996 (to appear).
30. M. Land and R. K. Belew, "No perfect two-state cellular automata for density classification exists," *Physical Review Letters* **74**, 5148 (1995).
31. M. S. Capcarrere, M. Sipper, and M. Tomassini, "A two-state, $r = 1$ cellular automaton that classifies density," *Physical Review Letters*, 1996 (to appear).
32. S. A. Kauffman, "Metabolic stability and epigenesis in randomly constructed genetic nets," *Journal of Theoretical Biology* **22**, 437 (1969).
33. S. Wolfram, "Statistical mechanics of cellular automata," *Reviews of Modern Physics* **55**, 601 (1983).