

## Designing cellular automata using a parallel evolutionary algorithm

Moshe Sipper<sup>a,\*</sup>, Marco Tomassini<sup>b</sup>, Mathieu S. Caparrere<sup>a</sup>

<sup>a</sup>Logic Systems Laboratory, Swiss Federal Institute of Technology, IN-Ecublens, CH-1015 Lausanne, Switzerland

<sup>b</sup>Logic Systems Laboratory, Swiss Federal Institute of Technology, and Computer Science Institute, University of Lausanne, CH-1015 Lausanne, Switzerland

### Abstract

We have previously shown that non-uniform Cellular Automata (CA) can be *evolved* to perform computational tasks, using the *cellular programming* evolutionary algorithm. In this paper we focus on two novel issues, namely the evolution of asynchronous CAs, and the fault tolerance of our evolved systems. We find that asynchrony presents a more difficult case for evolution though good CAs can still be attained. We show that our evolved systems exhibit graceful degradation in performance, able to tolerate a certain level of faults. Our motivation for this study stems in part by our desire to attain realistic systems that are more amenable to implementation as 'evolving ware', *evolware*.

PACS: 02.70.Rw; 07.05.Bx; 89.80. + h

Keywords: Non-uniform cellular automata; Asynchronous cellular automata; Cellular programming; Evolutionary computation; Fault tolerance; Evolware

### 1. Introduction

Cellular Automata (CA) are dynamical systems in which space and time are discrete. A cellular automaton 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. The state of a cell is determined by the previous states of a surrounding neighborhood of cells. This transition is usually specified in the form of a *rule table*, delineating the cell's next state for each possible neighborhood configuration. The cellular array (grid) is  $n$ -dimensional, where  $n = 1, 2, 3$  is used in practice (in this work we shall concentrate on  $n = 1$ ) [1,2].

CAs exhibit three notable features, namely massive parallelism, locality of cellular interactions, and simplicity of basic components (cells). They perform computations in a distributed fashion on a spatially extended grid; as such they differ from the standard approach to parallel

computation in which a problem is split into independent sub-problems, each solved by a different processor, later to be combined in order to yield the final solution. CAs suggest a new approach in which complex behavior arises in a bottom-up manner from non-linear, spatially extended, local interactions [3]. A major impediment preventing ubiquitous computing with CAs stems from the difficulty of utilizing their complex behavior to perform useful computations. Designing CAs to have a specific behavior or to perform a particular task is highly complicated, thus severely limiting their applications; automating the design (programming) process would greatly enhance the viability of CAs [3].

The model investigated in this paper is an extension of the CA model, termed *non-uniform cellular automata*. Such automata function in the same way as uniform ones, the only difference being in the cellular rules that need not be identical for all cells. We have previously shown that non-uniform CAs can be *evolved* to perform computational tasks, employing a local, co-evolutionary algorithm, an approach referred to as *cellular programming* (see Section 2). In this paper we focus on two extensions of our model: whereas previous studies were conducted using synchronous CAs, in Section 3 we study the evolution of *asynchronous*, non-uniform CAs

\* Corresponding author. Tel.: + 41 21 693 2658, fax: + 41 21 693 3705, e-mail: moshe.sipper@di.epfl.ch.

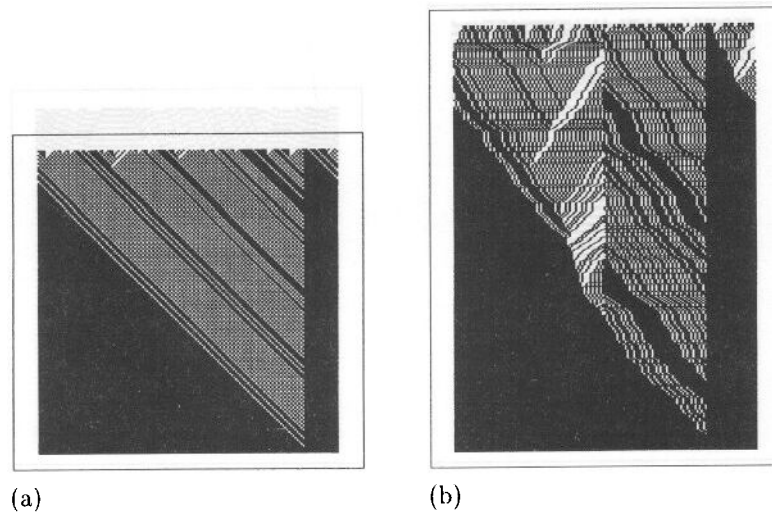


Fig. 1. One-dimensional density task: Operation of two co-evolved, non-uniform, connectivity radius  $r = 1$  CAs. White squares represent cells in state 0, black squares represent cells in state 1. The pattern of configurations is shown through time (which increases down the page). (a) A synchronous CA. Grid size is  $N = 149$ . CA is run for 150 time steps. (b) An asynchronous (model-1) CA. Grid size is  $N = 150$ , with two 75-cell blocks ( $\#_b = 2$ ). CA is run for 665 time steps. The randomly generated initial configurations have a density of 1s greater than 0.5, and the CAs relax to a fixed pattern of all 1s, which is the correct solution.

to perform two computational tasks. In Section 4 we investigate the resilience of our evolved CAs, namely how do they perform in the face of errors. Our conclusions are presented in Section 5. We believe that cellular programming holds potential for attaining 'evolving ware', *evolware*, which can be implemented in software, hardware, or other possible forms, such as *bioware*. Of particular interest is the issue of evolving hardware, which has recently made its appearance on the artificial evolution scene [4]. A prime motivation of the work reported in this paper stems from our desire to attain more realistic systems that are amenable to implementation as *evolware*.

## 2. Previous work

We had first studied non-uniform CAs in Refs. [5,6] and demonstrated in Ref. [7] that universal computation can be attained in such CAs. The evolution of non-uniform CAs was undertaken in Refs. [8–14], where the *cellular programming* algorithm was presented; we showed that high-performance, non-uniform CAs can be evolved to perform non-trivial, global computational tasks. The cellular programming algorithm is detailed in the above references; essentially, each cell of our non-uniform CA maintains an evolving "genome", encoding its rule table in the form of a bit string. The genomes are initialized at random, after which the CA is run on numerous randomly generated initial con-

figurations.<sup>1</sup> The cell's *fitness*, used to drive the evolutionary process, is its performance on these configurations, defined in accordance with the task at hand. We note in passing that our algorithm involves *local co-evolution*, as such differing from the standard genetic algorithm.

In Refs. [3,15] a standard genetic algorithm was used to evolve *uniform*, one-dimensional CAs to perform two computational tasks, density and synchronization. These experiments involved two-state CAs (i.e., each cell can be in one of two states, 0 or 1), with connectivity radius  $r = 3$ , meaning that each cell is connected to 3 neighbors on either side (thus, each cell has  $2r + 1$  neighbors including itself). Spatially periodic boundary conditions are used, resulting in a circular grid. We have used cellular programming to evolve *non-uniform* CAs with a minimal radius of  $r = 1$  to solve both these tasks (as well as others, including random number generation and image processing). The one-dimensional density task is to decide whether or not the initial configuration 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 (Fig. 1). In the one-dimensional synchronization task the CA, given any initial configuration, must reach a final

<sup>1</sup> The term "configuration" refers to an assignment of states to cells in the grid.

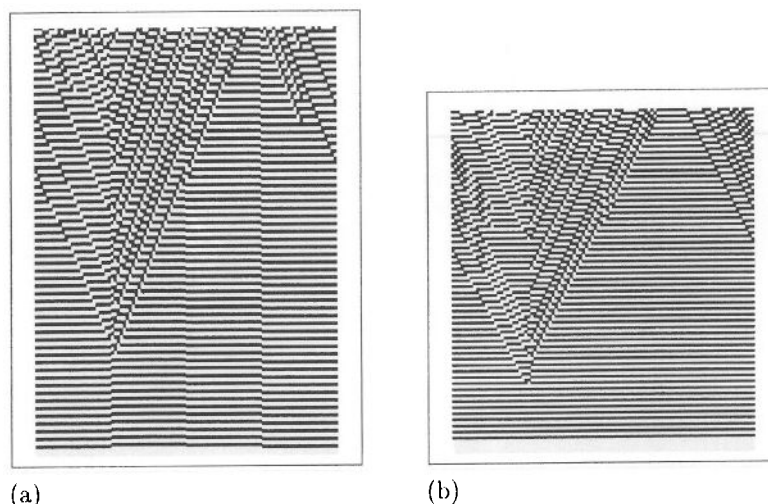


Fig. 2. One-dimensional synchronization task: Operation of a co-evolved, asynchronous (model-3), non-uniform CA, with connectivity radius  $r = 1$ . CA size is  $N = 150$ , partitioned into 4 blocks ( $\#_b = 4$ ) with updating order: block 1  $\rightarrow$  block 2  $\rightarrow$  block 0  $\rightarrow$  block 3  $\rightarrow$  block 1  $\rightarrow$  block 2  $\rightarrow$  ... Operation is shown for two randomly generated initial configurations. (a) The CA's configuration is depicted at every time step. (b) The CA's configuration is depicted at every logical step ( $= 4$  time steps).

configuration, within  $M$  time steps, that oscillates between all 0s and all 1s on successive time steps (Fig. 2). 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) [3,15].

Our previous studies involving cellular programming consisted of evolving *parallel cellular machines* to perform computational tasks. Our machine model was attained by considering a generalization of the original CA model, namely non-uniform CAs, where cellular rules need not necessarily be identical. In the following two sections we study two additional generalizations, namely asynchronous CAs and non-deterministic ones, our motivation stemming in part from our desire to attain more realistic systems that are amenable to implementation as *evolware*.

### 3. Evolving asynchronous CAs

One of the prominent features of the CA model is its synchronous mode of operation, meaning that all cells are updated simultaneously. A preliminary study of asynchronous CAs, where one cell is updated at each time step, was carried out in Ref. [16], where the different dynamical behavior of synchronous and asynchronous CAs was compared; they argued that some of the apparent self-organization of CAs is an artifact of the synchronization of the clocks. Ref. [17] noted that asynchronous updating makes it more difficult for information to propagate throughout the CA and that, furthermore, such CAs may be harder to analyze. Asynchronous CAs

have also been discussed in Refs. [6,18,19], though it seems clear that they have received a limited amount of attention to date.

The issue investigated in this section is that of evolving asynchronous CAs to perform the density and synchronization tasks. The grid is partitioned into *blocks* in which synchronous updating takes place (i.e., all cells within a block are updated simultaneously), while the blocks themselves are updated asynchronously (rather than have all blocks updated at once); thus, inter-block updating is synchronous while intra-block updating is asynchronous.<sup>2</sup> The number of blocks per grid,  $\#_b$ , is a tunable parameter, entailing a *scale* of asynchrony, ranging from complete synchrony ( $\#_b = 1$ ) to complete asynchrony ( $\#_b = N$ ). There are two main differences between our investigation and previous ones: (1) rather than consider only complete asynchrony ( $\#_b = N$ ), we have introduced the above scale, and (2) asynchronous CAs were previously studied from a more abstract point of view, whereas we are interested in *evolving* them to perform a veritable *computation*.

Three models of asynchrony are considered, which differ in the scheduling of intra-block updating (inter-block updating is always synchronous):

*Model 1:* At every time step each block is updated *independently* of the others with probability  $p_{\text{update}}$ .

<sup>2</sup> A preliminary investigation of a CA-derived model based on the "blocks" idea was carried out in Ref. [6].

chosen so as to insure that at least one block is updated per time step with probability  $\geq 0.99$ .

*Model 2:* Each time step a different block is chosen at random without replacement, such that every  $\#_b$  steps, all blocks are updated exactly once. We denote by *logical step* the succession of  $\#_b$  time steps necessary for one full update cycle, in which all cells are updated (thus, one logical step is equivalent to one time step in the synchronous model, with respect to cell updating).

*Model 3:* All blocks are updated in a fixed, random order every logical step. This is similar to the second model, in that each cell is guaranteed to have updated its state every logical step, however, the (random) update order is fixed (rather than selected anew each logical step). Note that though the update order is deterministic, this model is interesting in that cells are not updated in a regular manner; neighboring cells may be updated at different points in time, which renders the computation more difficult.

Cyclic behavior cannot arise in the first model since the notion of a logical step, i.e., a fixed number of time steps after which all cells will have been updated, does not exist; however, a fixed point, such as that desired for the density problem, can be attained. Models 2 and 3 can be applied to the synchronization problem since cyclic behavior may be attained, if one considers the CA's configuration every logical step, i.e., the alternation between all 0s and all 1s takes place every  $\#_b$  time steps.

Our results for the density task show that model-1 asynchronous CAs can be evolved whose performance is comparable to the synchronous case,<sup>3</sup> provided the number of blocks does not exceed three ( $\#_b \leq 3$ ); for  $\#_b > 3$ , successful asynchronous CAs did not evolve. Fig. 1b demonstrates the operation of an evolved, non-uniform, model-1 asynchronous CA on the density task. For the synchronization task, successful model-3 CAs with  $\#_b \leq 8$  were evolved (grid sizes considered were in the range  $N \in [100, 150]$ ); applying model 2, no successful CA had emerged from the evolutionary process. Fig. 2 demonstrates the operation of an evolved, non-uniform, model-3 asynchronous CA on the synchronization task.

The deterministic updating schedule of model 3 renders it easier for evolution to cope with, as compared with model 2. For both, however, an obstacle that hinders the evolutionary algorithm is the need to adapt to block boundaries. A "good" rule in cell  $i$  may be of no use, or even detrimental, in cell  $i + 1$ , if a block boundary

occurs between these two cells. Two strategies were observed to emerge from the evolutionary process in order to cope with this problem: either specialized rules are evolved at block boundaries (different than the rules present in the rest of the block), or a rule is evolved that is essentially insensitive to the presence or absence of a boundary.

#### 4. Fault tolerance in evolved CAs

In this section we return to *synchronous*, non-uniform CAs, our interest lying in studying their resilience, namely how do they perform in the face of errors. The CAs in question are those that have evolved to solve either the density or synchronization tasks, with our fault-tolerance investigation picking up upon *termination* of the evolutionary process. We shall focus on one type of error where a cell updates its state in a non-deterministic manner: at each time step, the cell's next state is that specified in the rule table, with probability  $1 - p_f$ , or the complementary one with probability  $p_f$ ;  $p_f$  is denoted the *fault probability*, representing the probability that a cell will incorrectly update its state. Fig. 3 depicts the operation of an evolved, non-uniform CA on the synchronization task for two different  $p_f$  values.

In order to study the evolved CAs' fault-tolerance properties, we have applied a number of measures, one of which is reported below. This measure, based on Ref. [20], involves a comparison between a "perfect" version of the CA ( $p_f = 0$ ) and a faulty one ( $p_f > 0$ ); both are presented with the same initial configuration at time step  $t = 0$ , and the Hamming distance between configurations at successive time steps is recorded.<sup>4</sup> This provides us with insight into the faulty CA's behavior, by measuring the amount by which it diverges from a "perfect" computation.

Our results, presented in Fig. 4, show that Hamming distance is a sigmoid-shaped function of the fault probability  $p_f$ . Essentially, three regions can be observed: 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. A more detailed investigation of this issue can be found in Ref. [21].

<sup>3</sup> Performance results for the synchronous case are reported, e.g., in Ref. [8].

<sup>4</sup> The Hamming distance between two configurations is the number of bits by which they differ.

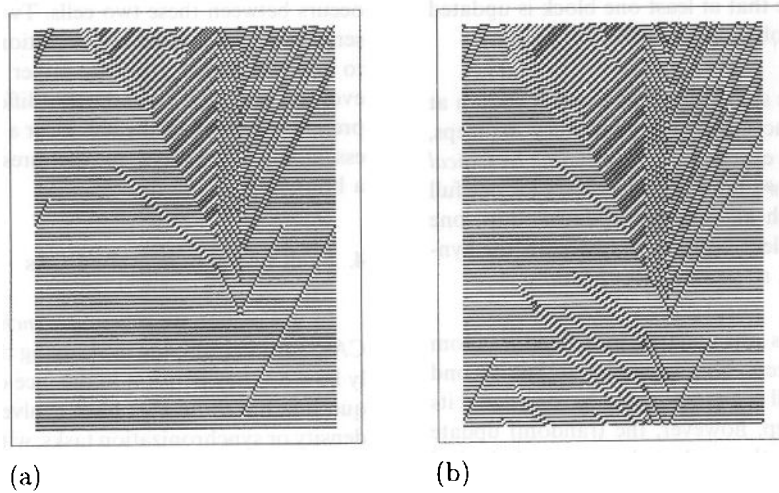


Fig. 3. One-dimensional synchronization task: Operation of a co-evolved, non-uniform,  $r = 1$  CA, with probability of fault  $p_f > 0$ . Grid size is  $N = 149$ . Initial configurations were generated at random: (a)  $p_f = 0.0001$ ; (b)  $p_f = 0.001$ .

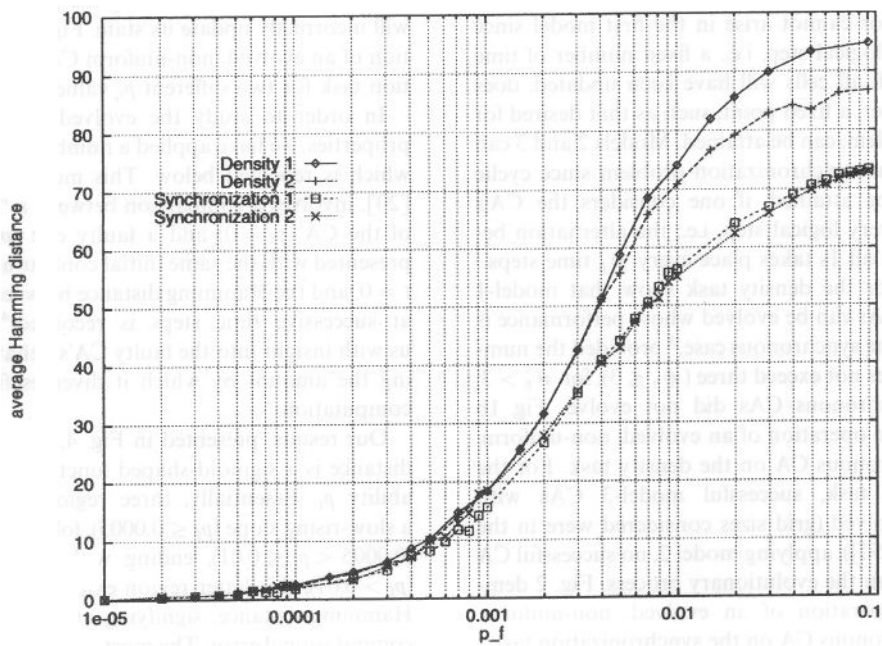


Fig. 4. Average Hamming distance versus fault probability  $p_f$ . Four CAs were studied – two that were evolved to solve the density task, and two that were evolved to solve the synchronization task. Grid size is  $N = 149$ . For each  $p_f$  value the CA under test was run on 1000 randomly generated 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.

## 5. Conclusions

We studied the evolution of non-uniform CAs via cellular programming, concentrating on two novel issues,

namely asynchrony and fault tolerance. We introduced three models of asynchrony, previously unstudied in this context, finding that asynchronous CAs can be evolved to perform the computational tasks in question. Though

it seems that asynchrony presents a more difficult case for evolution, it is premature to draw any definitive conclusions at this point, since we have only considered two problems, using relatively small-size grids. We feel that successful asynchronous CAs can be evolved, though this will probably entail larger grids (coupled with larger blocks). We found that our evolved systems exhibit graceful degradation in performance, able to tolerate a certain level of faults. Though preliminary, we hope that further studies along these lines will help deepen our knowledge of evolving cellular systems, as well as propel us toward the attainment of more realistic systems, that can ultimately be implemented as evolware.

## References

- [1] S. Wolfram, *Physica D* 10 (1984) 1.
- [2] T. Toffoli and N. Margolus, *Cellular Automata Machines* (The MIT Press, Cambridge, MA, 1987).
- [3] M. Mitchell, J.P. Crutchfield and P.T. Hraber, *Physica D* 75 (1994) 361.
- [4] E. Sanchez and M. Tomassini, (eds.), *Towards Evolvable Hardware*, Lecture Notes in Computer Science, Vol. 1062 (Springer, Berlin, 1996).
- [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, MA, 1994) 394–399.
- [6] M. Sipper, *Artificial Life J.* 2 (1995) 1.
- [7] M. Sipper, Quasi-uniform computation-universal cellular automata, in: *ECAL'95: 3rd European Conf. on Artificial Life*, eds. F. Morán, A. Moreno, J.J. Merelo and P. Chacón, Lecture Notes in Computer Science, Vol. 929 (Springer, Berlin, 1995) 544–554.
- [8] M. Sipper, *Physica D* 92 (1996) 193.
- [9] M. Sipper, Designing evolware by cellular programming, in: *Proc. Int. Conf. on Evolvable Systems: from Biology to Hardware (ICES96)*, Lecture Notes in Computer Science, (Springer, Heidelberg, 1997), to appear.
- [10] M. Sipper and E. Ruppín, *Physica D*, 99 (1997) 428.
- [11] M. Sipper and E. Ruppín, Co-evolving cellular architectures by cellular programming, in: *Proc. IEEE 3rd Int. Conf. on Evolutionary Computation (ICEC'96)*, (1996) 306–311.
- [12] M. Sipper and M. Tomassini, Co-evolving parallel random number generators, in: *Parallel Problem Solving from Nature - PPSN IV*, eds. H.-M. Voigt, W. Ebeling, I. Rechenberg and H.-P. Schwefel, Lecture Notes in Computer Science, Vol. 1141 (Springer, Heidelberg, 1996) 950–959.
- [13] M. Sipper and M. Tomassini, *Int. J. Modern Phys. C*, 7 (1996) 181.
- [14] M. Sipper, Evolving uniform and non-uniform cellular automata networks, in: *Annual Reviews of Computational Physics*, Vol. V, ed. D. Stauffer (World Scientific, 1997).
- [15] R. Das, J.P. Crutchfield, M. Mitchell and J.E. Hanson, Evolving globally synchronized cellular automata, in: *Proc. 6th Int. Conf. on Genetic Algorithms*, ed. L.J. Eshelman (Morgan Kaufmann, San Francisco, CA, 1995) 336–343.
- [16] T.E. Ingerson and R.L. Buvel, *Physica D* 10 (1984) 59.
- [17] S. Wolfram, *Physica D* 22 (1986) 385.
- [18] M.A. Nowak, S. Bonhoeffer and R.M. May, Spatial games and the maintenance of cooperation, *Proc. National Academy of Sciences USA*, Vol. 91 (1994) 4877–4881.
- [19] H. Bersini and V. Detour, Asynchrony induces stability in cellular automata based models, in: *Artificial Life IV*, eds. R.A. Brooks and P. Maes (The MIT Press, Cambridge, MA, 1994) 382–387.
- [20] S. Wolfram, *Rev. Modern Phys.*, 55 (1983) 601.
- [21] M. Sipper, M. Tomassini and O. Beuret, Studying probabilistic faults in evolved non-uniform cellular automata, *Int. J. Modern Physics C* 7(6) (1996) 923.