Proceedings of International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA97).

Evolving Asynchronous and Scalable Non-uniform Cellular Automata

Moshe Sipper, Marco Tomassini,* and Mathieu S. Capcarrere
Logic Systems Laboratory, Swiss Federal Institute of Technology, IN-Ecublens,
CH-1015 Lausanne, Switzerland. E-mail: {name.surname}@di.epfl.ch.

Abstract

We have previously shown that non-uniform cellular automata (CA) can be *evolved* to perform computational tasks, using the *cellular programming* algorithm. In this paper we focus on two novel issues, namely, the evolution of asynchronous CAs, and the scalability of evolved synchronous systems. We find that asynchrony presents a more difficult case for evolution though good CAs can still be attained. We describe an empirically-derived scaling procedure by which successful CAs of any size may be obtained from a particular evolved system. Our motivation for this study stems in part from our desire to attain realistic systems that are more amenable to implementation as "evolving ware," *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 at the next time step 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 $n = 1$).

CAs exhibit three notable features, namely, massive parallelism, locality of cellular interactions, and simplicity of basic components (cells). A major impediment preventing ubiquitous computing with CAs stems from the difficulty of utilizing their complex behavior to perform useful computations. Designing CAs to exhibit a specific behavior or perform a particular task is highly complicated, thus severely limiting their applications; automating the design (programming) process would greatly enhance the viability of CAs [1].

One possible approach, taken in this paper, is to employ artificial evolution. The model investigated by us is an extension of the original CA model, termed *non-uniform cellular automata*. Such automata function in the same way as uniform ones, the only difference being that the local cellular rules need not be identical for all cells. Our approach involves the *cellular programming* algorithm, by which non-uniform CAs evolve to perform non-trivial, global computational tasks [2–7] (for a description of the algorithm the reader is referred to these references).

The tasks for which non-uniform CAs were evolved via cellular programming include, among others, density and synchronization. Both involve two-state CAs, i.e., each cell can be in one of two states, 0 or 1, with connectivity radius $r = 1$, meaning that each cell is connected to one neighbor on either side (thus, each cell has $2r + 1$ neighbors, including itself). 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 (Figure 1) [1, 2]. The term 'configuration' refers to an assignment of states to grid cells. 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 (Figure 2) [2, 8]. Spatially periodic boundary conditions are applied, resulting in a circular grid (for an $r = 1$ CA this means that the leftmost and rightmost cells are connected). It should be emphasized that both tasks comprise non-trivial computational prob-

*M. Tomassini is also with the Computer Science Institute, University of Lausanne.

lems for a small radius CA ($r \ll N$, where N is the grid size) [1, 2, 8].

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 this paper we study an additional generalization, namely, asynchronous CAs, as well as the scalability of evolved synchronous systems. Our motivation stems in part from our desire to attain more realistic systems that are amenable to implementation as "evolving ware," *evolware* [2, 3].

2 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. [9], where the different dynamical behavior of synchronous and asynchronous CAs was compared; the authors argued that some of the apparent self-organization of CAs is an artifact of the synchronization of the clocks. Ref. [10] noted that asynchronous updating makes it more difficult for information to propagate through the CA and that, furthermore, such CAs may be harder to analyze. Asynchronous CAs have also been discussed in Refs. [2, 11, 12], 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. 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_{update}, chosen so as to insure that at least one block is updated per time step with probability ≥ 0.99.

Model 2 At 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,[1] provided the number of blocks does not exceed three ($\#_b \leq 3$); for $\#_b > 3$, successful asynchronous CAs did not evolve. Figure 1 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.

[1] Performance results for the synchronous case are reported, e.g., in Refs. [2, 5].

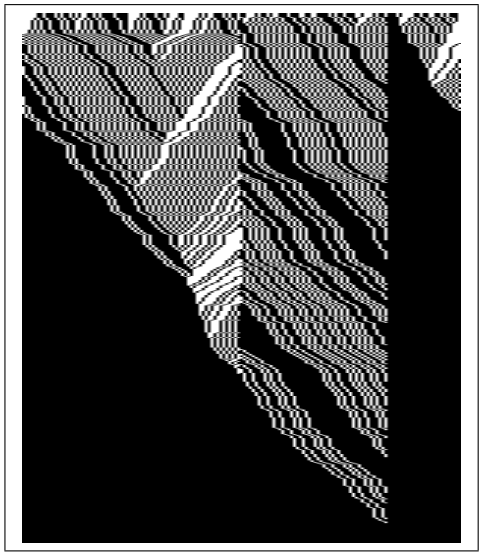

Figure 1: One-dimensional density task. Operation of a coevolved, non-uniform CA with connectivity radius $r = 1$. The CA is asynchronous, model 1. Grid size is $N = 150$, with two 75-cell blocks ($\#_b = 2$). White squares represent cells in state 0, black squares represent cells in state 1. The pattern of configurations is shown for the first 665 time steps, with time increasing down the page. The randomly generated initial configuration has a density of 1s greater than 0.5, and the CA relaxes to a fixed pattern of all 1s, which is the correct output.

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 through 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.

3 Scaling evolved CAs

In this section we return to *synchronous*, non-uniform CAs, our interest lying in the scalability issue. Essentially, this involves two separate matters: the evolutionary algorithm and the evolved solutions. As to the former, we note that as our cellular programming algorithm is local it scales better in terms of hardware resources than the standard (global) genetic algorithm; adding grid cells requires only local connections in our case whereas the standard genetic algorithm includes global operators such as fitness ranking and crossover [2]. In this section we concentrate on the second issue, namely, how can the grid size be modified given an evolved grid of a particular length, i.e., how can evolved solutions be scaled? This has been purported as an advantage of uniform CAs, since one can directly use the evolved rule in a grid of any desired size. However, this form of *simple* scaling does not bring about *task* scaling; as demonstrated, e.g., in Ref. [13] for the density task, performance decreases as grid size increases. Previously, we had attained successful systems for a random number generation task using a simple scaling scheme involving the duplication of the rules grid [7]. Below we report on a more sophisticated, empirically-obtained scheme that has proven successful.

Given an evolved non-uniform CA of size N, our goal is to obtain a grid of size N', where N' is given but arbitrary (N' may be $> N$ or $< N$), such that the original performance level is maintained. This requires an algorithm for determining which rule should be placed in each cell of the size N' grid, so as to preserve the original grid's "essence," i.e., its emergent global behavior. Thus, we must determine what characterizes this latter behavior. We first note that there are two basic rule structures of importance in the original grid (shown for $r = 1$):

- The *local structure* with respect to cell i, $i \in \{0, \ldots, N-1\}$, is the set of three rules in cells $i-1$, i, and $i+1$ (indices are computed modulus N since the grid is circular).
- The *global structure* is derived by observing the *zones* of identical rules present in the grid.[2] For example, for the following evolved $N = 15$ grid:

$R_1R_1R_1R_1$	R_2R_2	R_3	$R_4R_4R_4R_4$	R_1	$R_5R_5R_5$

  where R_j, $j \in \{1, \ldots, 5\}$, denotes a distinct rule, the number of zones is 6, and the global structure is given by the list $\{R_1, R_2, R_3, R_4, R_1, R_5\}$.

We have found that if these structures are preserved, the scaled CA's behavior is identical to that of the original one. A heuristic principle is to expand (or reduce) a zone of identical rules which spans at least four cells, while keeping intact zones of length three or less. It is straightforward to observe that a zone of length one or two should be left untouched, so as to maintain the local structure. As for a zone of length three, there is no a priori reason why it should

[2] We use here the term 'zone' to denote a region of cells with the same evolved rule; this is not to be confused with the synchrony 'blocks' of the previous section.

be left unperturbed, rather, this has been found to hold empirically. A possible explanation may be that in such a three-cell zone the local structure $R_jR_jR_j$ appears only once, thereby comprising a "primitive" unit that must be maintained. As an example of this procedure, consider the above $N = 15$ CA– scaling this grid to size $N' = 19$ results in:

$R_1R_1R_1R_1R_1$	R_2R_2	R_3	$R_4R_4R_4R_4R_4R_4R_4$	R_1	$R_5R_5R_5$

Note that both the local and global structures are preserved. We tested our scaling procedure on several CAs that were evolved to solve the synchronization task. The original grid sizes were $N = 100, 150$, which were then scaled to grids of sizes $N' = 200, 300, 350, 450, 500, 750$. In all cases the scaled grids exhibited the same performance level as that of the original ones. An example of a scaled system is shown in Figure 2.

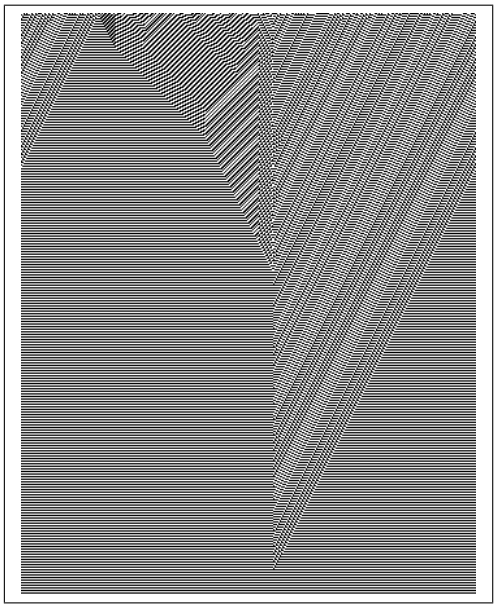

Figure 2: One-dimensional synchronization task– example of a scaled CA. An evolved, size $N = 149$ CA was scaled to a size $N' = 350$ CA, shown above.

4 Conclusions

We studied the evolution of non-uniform CAs via cellular programming, concentrating on two novel issues, namely, asynchrony and scalability. 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 next described a scaling procedure for synchronous CAs, by which an evolved system of given size may be used to obtain augmented or reduced grids. Our tests suggest that this procedure yields scaled systems whose performance level is identical to the original one. 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 adaptive systems, that can ultimately be implemented as evolving ware, evolware.

References

[1] M. Mitchell, J. P. Crutchfield, and P. T. Hraber. Evolving cellular automata to perform computations: Mechanisms and impediments. *Physica D*, 75:361–391, 1994.

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

[3] M. Sipper. The evolution of parallel cellular machines: Toward evolware. *BioSystems*, 1997. (to appear).

[4] M. Sipper. Evolving uniform and non-uniform cellular automata networks. In D. Stauffer, editor, *Annual Reviews of Computational Physics*, volume V. World Scientific, Singapore, 1997. (to appear).

[5] M. Sipper. Co-evolving non-uniform cellular automata to perform computations. *Physica D*, 92:193–208, 1996.

[6] M. Sipper and E. Ruppin. Co-evolving architectures for cellular machines. *Physica D*, 99:428–441, 1997.

[7] M. Sipper and M. Tomassini. Generating parallel random number generators by cellular programming. *International Journal of Modern Physics C*, 7(2):181–190, 1996.

[8] R. Das, J. P. Crutchfield, M. Mitchell, and J. E. Hanson. Evolving globally synchronized cellular automata. In L. J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 336–343, San Francisco, CA, 1995. Morgan Kaufmann.

[9] T. E. Ingerson and R. L. Buvel. Structure in asynchronous cellular automata. *Physica D*, 10:59–68, 1984.

[10] S. Wolfram. Approaches to complexity engineering. *Physica D*, 22:385–399, October 1986.

[11] M. A. Nowak, S. Bonhoeffer, and R. M. May. Spatial games and the maintenance of cooperation. *Proceedings of the National Academy of Sciences USA*, 91:4877–4881, May 1994.

[12] H. Bersini and V. Detour. Asynchrony induces stability in cellular automata based models. In R. A. Brooks and P. Maes, editors, *Artificial Life IV*, pages 382–387, Cambridge, Massachusetts, 1994. The MIT Press.

[13] J. P. Crutchfield and M. Mitchell. The evolution of emergent computation. *Proceedings of the National Academy of Sciences USA*, 92(23):10742–10746, 1995.