Evolware

Designing Evolware by Cellular Programming

Moshe Sipper

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

Abstract. A major impediment preventing ubiquitous computing with cellular automata (CA) stems from the difficulty of utilizing their complex behavior to perform useful computations. In this paper *non-uniform* CAs are studied, presenting the *cellular programming* algorithm for co-evolving such systems to perform computations. The algorithm is applied to five computational tasks: density, synchronization, ordering, boundary computation, and thinning; our results show that non-uniform CAs can attain high computational performance, and furthermore, that such systems can be evolved rather than designed. 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. We have recently implemented an evolving, online, autonomous hardware system based on the approach described herein.

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 [34, 30].

CAs exhibit three notable features: 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 [14].

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 perform a particular task is highly complicated, thus severely limiting their applications; automating the design (programming) process would greatly enhance the viability of CAs [14]. A prime motivation for studying CAs stems from the observation that they are naturally suited for hardware implementation, with the potential of exhibiting extremely fast and reliable computation that is robust to noisy input data and component failure [7].

The model investigated in this paper is an extension of the CA model, termed *non-uniform cellular automata* [20]. 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. Our main focus is on the *evolution* of non-uniform CAs to perform computational tasks, employing a local, co-evolutionary algorithm, an approach referred to as *cellular programming.* 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 [19]. We have recently implemented an evolving, online, autonomous hardware system based on the approach described herein [8].

Our aim in this paper is to introduce the cellular programming approach, toward which end we shall delineate the basic methodology and present examples of co-evolved, non-uniform CAs. In Section 2 we present previous work on non-uniform CAs and evolving CAs. The cellular programming algorithm is delineated in Section 3, and applied to five computational tasks in Section 4: density, synchronization, ordering, boundary computation, and thinning. We demonstrate that high performance is attained on these tasks using one-dimensional grids as well as previously unstudied two-dimensional ones. Our findings are discussed in Section 5.

2 Previous work

The application of genetic algorithms to the evolution of *uniform* cellular automata was initially studied by [16] and recently undertaken by the EVCA (evolving CA) group [15, 14, 13, 6, 5]. They carried out experiments involving uniform, one-dimensional CAs with $k = 2$ and $r = 3$, where k denotes the number of possible states per cell and r denotes the radius of a cell, i.e., the number of neighbors on either side (thus, each cell has $2r + 1$ neighbors, including itself). Spatially periodic boundary conditions are used, resulting in a circular grid. A common method of examining the behavior of one-dimensional CAs 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). The term 'configuration' refers to an assignment of 1 states to several cells, and 0s otherwise.

The EVCA group employed a standard genetic algorithm to evolve uniform CAs to perform two computational tasks, namely density and synchronization (see Section 4). The algorithm uses a randomly generated initial population of CAs with $k = 2$, $r = 3$. Each CA is represented by a bit string, delineating its rule table, containing the output bits for all possible neighborhood configurations (i.e., the bit at position 0 is the state to which neighborhood configuration 0000000 is mapped to and so on until bit 127 corresponding to neighborhood configuration 1111111). The bit string, known as the "genome", is of size $2^{2r+1} = 128$, resulting in a huge search space of size 2^{128}. Each CA in the population was run for a maximum number of M time steps, after which its fitness was evaluated, defined as the fraction of cell states correct at the last time step. Using the genetic algorithm highly successful CA rules were found for both tasks [15, 14, 5].

The model investigated in this paper is that of non-uniform CAs, where cellular rules need not be identical for all cells. As noted in Section 1, CAs lend themselves naturally to hardware implementation, which is one of the primary motivations for their study. Note that from a hardware point of view the resources required by non-uniform CAs are identical to those of uniform ones since a cell in both cases contains a rule (albeit not necessarily the same one in our case). A prime motivation for studying non-uniform CAs stems from the observation that the uniform model is essentially "programmed" at an extremely low-level [18]. A *single* rule is sought that must be applied universally to all cells in the grid, a task which may be arduous even for evolutionary approaches. For non-uniform CAs search space sizes are vastly larger than with uniform CAs, a fact that initially seems as an impediment; however, we have found that this model presents novel dynamics, offering new and interesting paths in the evolution of complex systems.

We have previously applied the non-uniform CA model to the investigation of artificial life issues, presenting multi-cellular "organisms" that display several interesting behaviors, including reproduction, growth and mobility. We also studied evolution in the context of various environments, observing genotypic as well as phenotypic effects [20, 23, 21]. In [22, 25] we examined the issue of universal computation in two-dimensional CAs. We demonstrated that universality can be attained in non-uniform, 2-state, 5-neighbor cellular space (i.e., with a minimal number of states as well as a minimal neighborhood), which is not universal in the uniform case [2]. The universal systems we presented are simpler than previous ones and are *quasi*-uniform, meaning that the number of distinct rules is extremely small with respect to rule space size; furthermore, the rules are distributed such that a subset of dominant rules occupies most of the grid [22, 25, 24].

The co-evolution of non-uniform, one-dimensional CAs to perform computations was undertaken in [24, 25]. In [24] we presented results pertaining to the density task, showing that high performance, non-uniform CAs can be co-evolved not only with radius $r = 3$, as studied by [15, 14], but also for smaller radiuses, most notably $r = 1$ which is minimal. In [25] we showed that high performance can be attained for the synchronization task as well, with $r = 1$, using the cellular programming algorithm. It was also found that evolved systems exhibiting high performance are quasi-uniform.

The one-dimensional density and synchronization tasks are elaborated upon in Section 4, along with recent results pertaining to two-dimensional grids and novel tasks. Our main findings from our previous work, as well as that reported in this paper are:

1. Universal computation can be attained in simple, non-uniform cellular spaces that are not universal in the uniform case. This is accomplished by utilizing a small number of different rules (quasi-uniformity).

2. Non-uniform CAs can attain high performance on non-trivial computational tasks.
3. Non-uniform CAs can be co-evolved to perform computations, with high performance systems exhibiting quasi-uniformity.
4. Non-uniformity may reduce connectivity requirements, i.e., the use of smaller cellular neighborhoods is made possible.

3 The cellular programming algorithm

We study 2-state, non-uniform CAs, in which each cell may contain a different rule. A cell's rule table is encoded as a bit string, known as the "genome", containing the output bits for all possible neighborhood configurations (as in Section 2). Rather than employ a *population* of evolving, uniform CAs, as with genetic algorithm approaches, our algorithm involves a *single*, non-uniform CA of size N, with cell rules initialized at random. Initial configurations are then generated at random, in accordance with the task at hand. For each initial configuration the CA is run for M time steps. Each cell's *fitness* is accumulated over $C = 300$ initial configurations, where a single run's score is 1 if the cell is in the correct state after M iterations, and 0 otherwise. After every C configurations evolution of rules occurs by applying crossover and mutation. This evolutionary process is performed in a completely *local* manner, where genetic operators are applied only between directly connected cells. It is driven by $nf_i(c)$, the number of fitter neighbors of cell i after c configurations. The pseudo-code of our algorithm is delineated in Figure 1.

Crossover between two rules is performed by selecting at random (with uniform probability) a single crossover point and creating a new rule by combining the first rule's bit string before the crossover point with the second rule's bit string from this point onward. Mutation is applied to the bit string of a rule with probability 0.001 per bit.

There are two main differences between our algorithm and the standard genetic algorithm: (a) A standard genetic algorithm involves a population of evolving, uniform CAs; all CAs are *ranked* according to fitness, with crossover occurring between *any* two individuals in the population. Thus, while the CA runs in accordance with a local rule, evolution proceeds in a *global* manner. In contrast, our algorithm proceeds *locally* in the sense that each cell has access only to its locale, not only during the run but also during the evolutionary phase, and no global fitness ranking is performed. (b) The standard genetic algorithm involves a population of *independent* problem solutions; the CAs in the population are assigned fitness values independent of one another, and interact only through the genetic operators in order to produce the next generation. In contrast, our CA *co-evolves* since each cell's fitness depends upon its evolving neighbors.

This latter point comprises a prime difference between our algorithm and parallel genetic algorithms, which have attracted attention over the past few years. These aim to exploit the inherent parallelism of evolutionary algorithms,

```
for each cell $i$ in CA do in parallel
    initialize rule table of cell $i$
    $f_i = 0$ { fitness value }
end parallel for
$c = 0$ { initial configurations counter }
while not done do
    generate a random initial configuration
    run CA on initial configuration for $M$ time steps
    for each cell $i$ do in parallel
        if cell $i$ is in the correct final state then
            $f_i = f_i + 1$
        end if
    end parallel for
    $c = c + 1$
    if $c$ mod $C = 0$ then { evolve every $C$ configurations}
        for each cell $i$ do in parallel
            compute $nf_i(c)$ { number of fitter neighbors }
            if $nf_i(c) = 0$ then rule $i$ is left unchanged
            else if $nf_i(c) = 1$ then replace rule $i$ with the fitter neighboring rule,
                                  followed by mutation
            else if $nf_i(c) = 2$ then replace rule $i$ with the crossover of the two fitter
                                  neighboring rules, followed by mutation
            else if $nf_i(c) > 2$ then replace rule $i$ with the crossover of two randomly
                                  chosen fitter neighboring rules, followed by mutation
            end if
            $f_i = 0$
        end parallel for
    end if
end while
```

Fig. 1. Pseudo-code of the cellular programming algorithm.

thereby decreasing computation time and enhancing performance [32]. A number of models have been suggested, among them coarse-grained, island models [28, 3, 29], and fine-grained, grid models [31, 12]. The latter resemble our system in that they are massively parallel and local; however, the co-evolutionary aspect is missing. As we wish to attain a system displaying global computation, the individual cells do not evolve independently as with genetic algorithms (be they parallel or serial), i.e., in a "loosely-coupled" manner, but rather co-evolve, thereby comprising a "tightly-coupled" system.

4 Results

In this section we study five computational tasks using one-dimensional grids as well as previously unstudied two-dimensional ones: density (Section 4.1), synchronization (Section 4.2), ordering (Section 4.3), rectangle-boundary (Section 4.4), and thinning (Section 4.5). Minimal cellular spaces are used: 2-state, $r = 1$ for the one-dimensional case and 2-state, 5-neighbor for the two-dimensional one. The total number of initial configurations per evolutionary run was in the range $[10^5, 10^6]$. Performance values reported hereafter represent the average fitness of all grid cells after C configurations, normalized to the range $[0, 1]$; these are obtained during execution of the cellular programming algorithm.

4.1 The density task

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. As noted by [14], the density task comprises a non-trivial computation for a small radius CA ($r \ll N$, where N is the grid size); the density is a global property of a configuration whereas a small-radius CA relies solely on local interactions. Since the 1s can be distributed throughout the grid, propagation of information must occur over large distances (i.e., $O(N)$). The minimum amount of memory required for the task is $O(\log N)$ using a serial scan algorithm, thus the computation involved corresponds to recognition of a non-regular language.

We have studied this task in [24, 25] using non-uniform, one-dimensional, minimal radius $r = 1$ CAs of size $N = 149$. The search space involved is extremely large; since each cell contains one of 2^8 possible rules this space is of size $(2^8)^{149} = 2^{1192}$. In contrast, the size of *uniform*, $r = 1$ CA rule space is small, consisting of only $2^8 = 256$ rules. This enabled us to test each and every one of these rules, a feat not possible for larger values of r, finding that the maximal performance of uniform, $r = 1$ CAs on the density task is 0.83.

For the cellular programming algorithm we used randomly generated initial configurations, uniformly distributed over densities in the range $[0, 1]$, with the CA being run for $M = 150$ time steps (thus, computation time is linear with grid size). We found that non-uniform CAs had co-evolved that exhibit peak performance values as high as 0.93; furthermore, these consist of a grid in which one rule dominates, a situation referred to as quasi-uniformity (Section 2). Figure 2 demonstrates the operation of one such co-evolved CA along with a rules map, depicting the distribution of rules by assigning a unique color to each distinct rule.

The results obtained by us using a minimal radius of $r = 1$ are comparable to those of [15], who used uniform CAs of radius $r = 3$; furthermore, we attain notably higher performance than any *possible* uniform, $r = 1$ CA. This suggests that non-uniformity reduces connectivity requirements, i.e., the use of smaller cellular neighborhoods is made possible. A detailed investigation of the one-dimensional density task can be found in [24, 25].

The density task can be extended in a straightforward manner to two-dimensional grids. Applying our algorithm to such grids yielded notably higher performance than the one-dimensional case, with peak values of 0.99. Figure 3 demonstrates the operation of one such co-evolved CA. Qualitatively, we observe the CA's

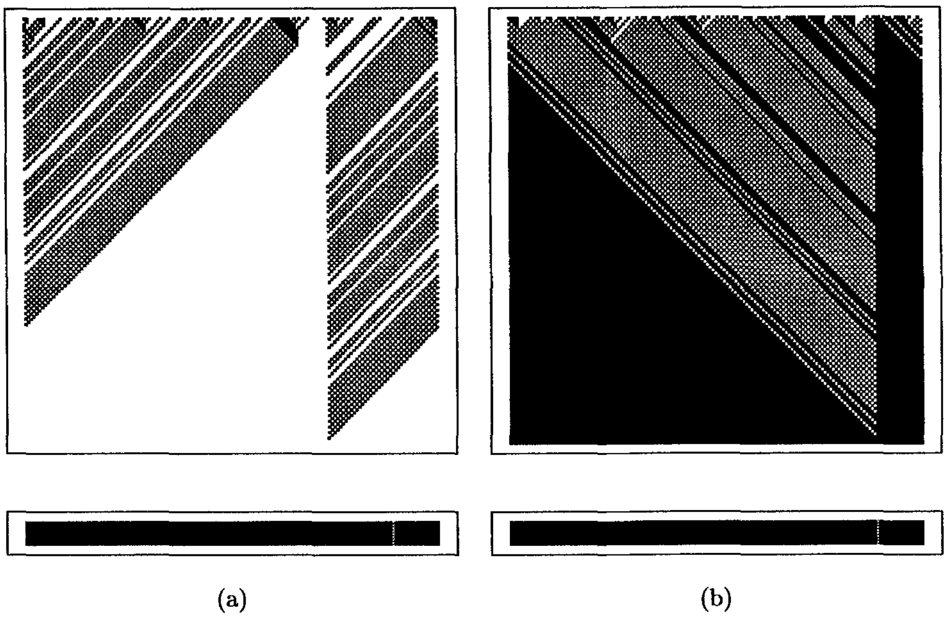
Fig. 2. One-dimensional density task: Operation of a co-evolved, non-uniform, connectivity radius $r = 1$ CA. Grid size is $N = 149$. White squares represent cells in state 0, black squares represent cells in state 1. The pattern of configurations is shown through time (which increases down the page). Top figures depict space-time diagrams, bottom figures depict rule maps. Initial configurations were randomly generated. (a) Initial density of 1s is 0.40, final density is 0. (b) Initial density of 1s is 0.60, final density is 1. The CA relaxes in both cases to a fixed pattern of all 0s or all 1s, correctly classifying the initial configuration.

"strategy" of successively classifying local densities, with the locality range increasing over time; "competing" regions of density 0 and density 1 are manifest, ultimately relaxing to the correct fixed point.

4.2 The synchronization task

The one-dimensional synchronization task was introduced by [5] and studied by us in [25] using non-uniform CAs. In this 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. This task comprises a non-trivial computation for a small radius CA; it belongs to a class of problems studied in other domains, such as distributed computing, known as firing squad problems [11].

In [25] we studied non-uniform, one-dimensional, minimal radius $r = 1$ CAs of size $N = 149$. As for the density task, we first tested all possible uniform, $r = 1$ CAs on the synchronization task, finding that the maximal performance is 0.84. For the cellular programming algorithm we used randomly generated initial

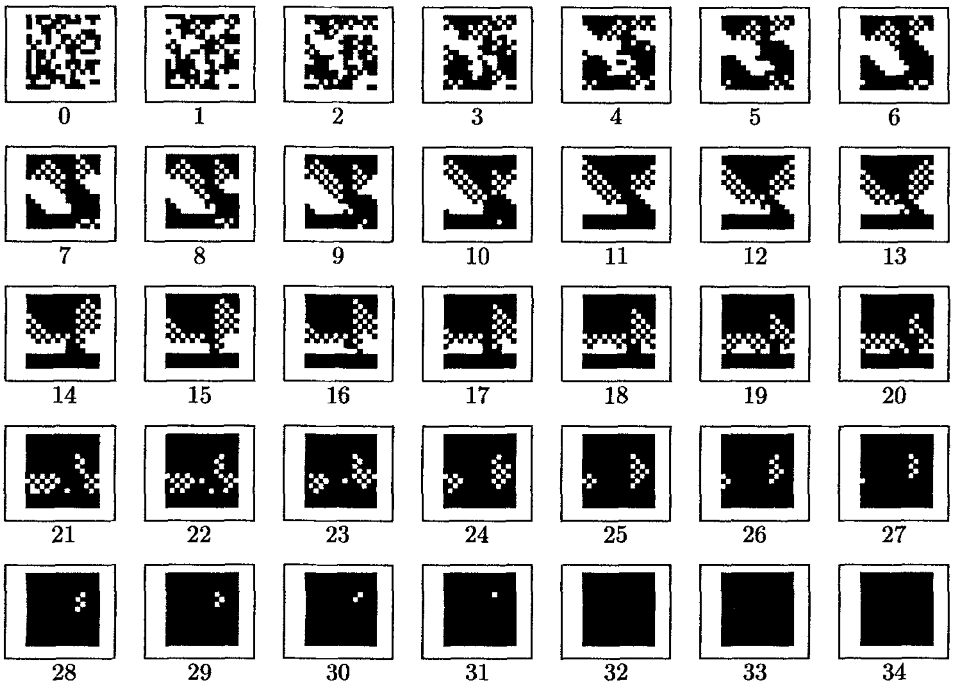

Fig. 3. Two-dimensional density task: Operation of a co-evolved, non-uniform, 2-state, 5-neighbor CA. Grid size is $N = 225$ (15×15). Initial density of 1s is 0.51, final density is 1. Numbers at bottom of images denote time steps.

configurations, uniformly distributed over densities in the range $[0, 1]$, with the CA being run for $M = 150$ time steps. We found that quasi-uniform CAs had co-evolved that exhibit perfect performance, thereby surpassing any possible uniform CA. Figure 4 depicts the operation of two such co-evolved CAs, along with rule maps. A detailed investigation of the one-dimensional synchronization task can be found in [25]. This task can also be extended in a straightforward manner to two-dimensional grids, an investigation of which we have carried out; our results show that perfect performance can be co-evolved for such CAs as well.

4.3 The ordering task

In this task, the one-dimensional CA, given any initial configuration, must reach a final configuration in which all 0s are placed on the left side of the grid and all 1s on the right side. The ordering task may be viewed as a variant of the density task and is clearly non-trivial using similar arguments to those of Section 4.1. It is interesting in that the output is not a uniform configuration of all 0s or all 1s as with the density and synchronization tasks.

Testing all uniform, $r = 1$ CAs on the ordering task we found that the

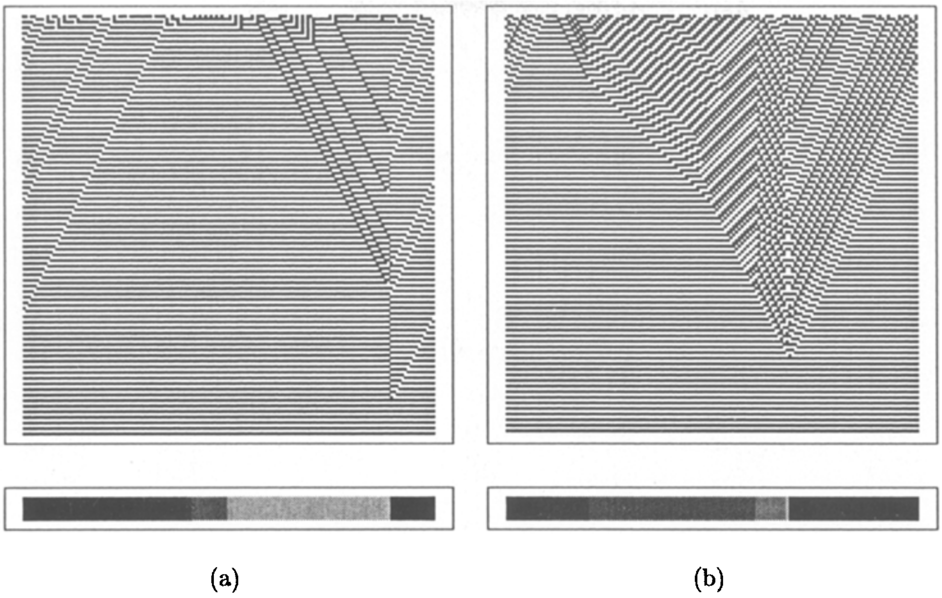

Fig. 4. One-dimensional synchronization task: Operation of two co-evolved, non-uniform, $r = 1$ CAs. Grid size is $N = 149$. Top figures depict space-time diagrams, bottom figures depict rule maps.

maximal performance is 0.71. Our algorithm yielded quasi-uniform CAs with fitness values as high as 0.93, one of which is depicted in Figure 5. As with the previous two tasks we find that non-uniform CAs can be co-evolved to attain high performance, exceeding that of the best uniform CA.

4.4 The rectangle-boundary task

The possibility of applying CAs to perform image processing tasks arises as a natural consequence of their architecture; in a two-dimensional CA, a cell (or a group of cells) can correspond to an image pixel, with the CA's dynamics designed so as to perform a desired image processing task. Earlier work in this area, carried out mostly in the 1960s and the 1970s, was treated in [17], with more recent applications presented in [1, 10].

The final two tasks we study involve image processing operations. In this section we discuss a two-dimensional boundary computation: given an initial configuration consisting of a non-filled rectangle, the CA must reach a final configuration in which the rectangular region is filled, i.e., all cells within the confines of the rectangle are in state 1, and all other cells are in state 0. Initial configurations consist of random-sized rectangles placed randomly on the grid (in our simulations, cells within the rectangle in the initial configuration were set to state 1 with probability 0.3; cells outside the rectangle were set to 0). Note that boundary cells can also be absent in the initial configuration. This operation can

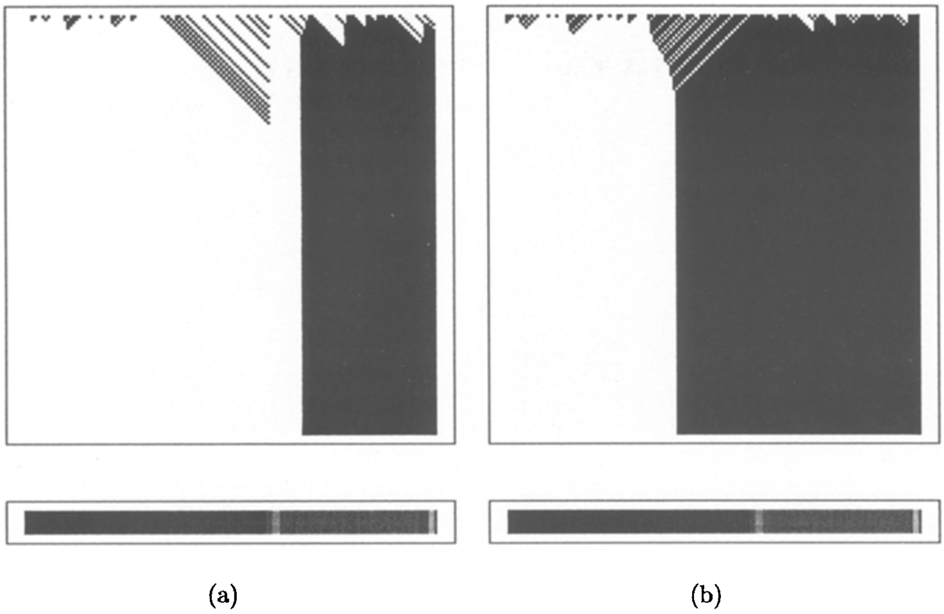

Fig. 5. One-dimensional ordering task: Operation of a co-evolved, non-uniform, $r = 1$ CA. Top figures depict space-time diagrams, bottom figures depict rule maps. (a) Initial density of 1s is 0.315, final density is 0.328. (b) Initial density of 1s is 0.60, final density is 0.59.

be considered a form of image enhancement, used, e.g., for treating corrupted images. Using cellular programming, non-uniform CAs were evolved with peak performance values of 0.99, one of which is depicted in Figure 6.

Upon studying the (two-dimensional) rules map of the co-evolved, non-uniform CA, we found that the grid is quasi-uniform, with one dominant rule present in most cells. This rule maps the cell's state to zero if the number of neighboring cells in state 1 (including the cell itself) is less than two, otherwise mapping the cell's state to one[1]. Thus, growing regions of 1s are more likely to occur within the rectangle confines than without.

4.5 The thinning task

Thinning (also known as skeletonization) is a fundamental preprocessing step in many image processing and pattern recognition algorithms. When the image consists of strokes or curves of varying thickness it is usually desirable to reduce them to thin representations located along the approximate middle of the original figure. Such "thinned" representations are typically easier to process in later stages, entailing savings in both time and storage space [9].

[1] This is referred to as a totalistic rule, in which the state of a cell depends only on the sum of the states of its neighbors at the previous time step, and not on their individual states [33].

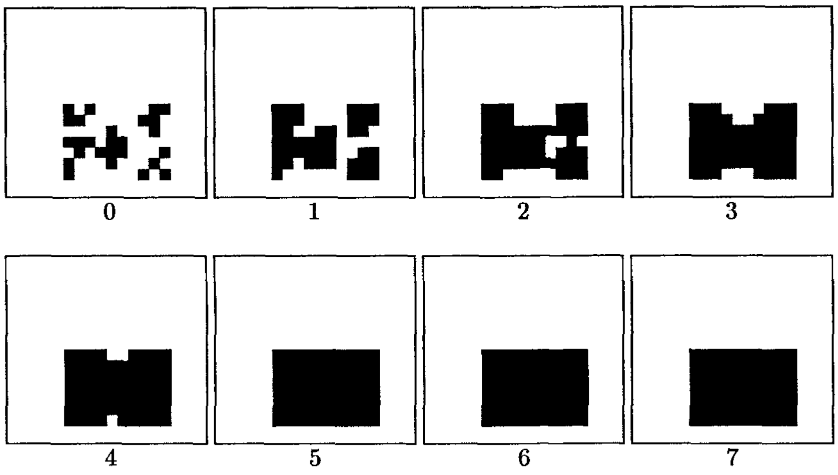

Fig. 6. Two-dimensional rectangle-boundary task: Operation of a co-evolved, non-uniform, 2-state, 5-neighbor CA. Grid size is $N = 225$ (15×15). Numbers at bottom of images denote time steps.

While the first thinning algorithms were designed for serial implementation, current interest lies in parallel systems, early examples of which were presented in [17]. The difficulty of designing a good thinning algorithm using a small, local cellular neighborhood, coupled with the task's importance has motivated us to explore the possibility of applying the cellular programming algorithm.

In [9] four sets of binary images were considered, two of which consist of rectangular patterns oriented at different angles. The algorithms presented therein employ a two-dimensional grid with a 9-cell neighborhood; each parallel step consists of two sub-iterations in which distinct operations take place. The set of images considered by us consists of rectangular patterns oriented either horizontally or vertically; while more restrictive than that of [9], it is noted that we employ a smaller neighborhood (5-cell) and do not apply any sub-iterations.

Figure 7 demonstrates the operation of a co-evolved CA performing the thinning task. Although the evolved grid does not compute perfect solutions, we observe, nonetheless, good thinning "behavior" upon presentation of rectangular patterns as defined above (Figure 7a); furthermore, partial success is demonstrated when presented with more difficult images involving intersecting lines (Figure 7b).

5 Discussion

A major impediment preventing ubiquitous computing with CAs stems from the difficulty of utilizing their complex behavior to perform useful computations. We presented the cellular programming algorithm for co-evolving computation in

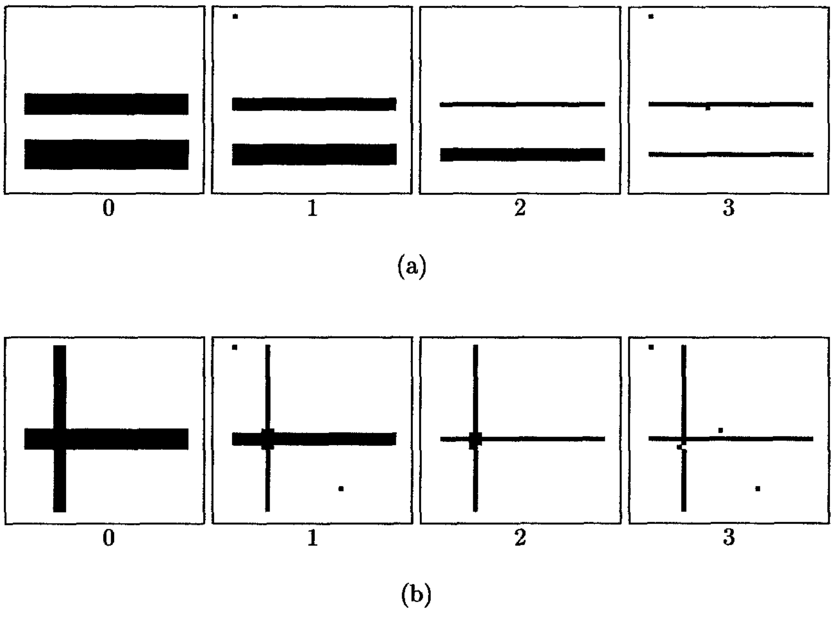

Fig. 7. Two-dimensional thinning task: Operation of a co-evolved, non-uniform, 2-state, 5-neighbor CA. Grid size is $N = 1600$ (40×40). Numbers at bottom of images denote time steps. (a) Two separate lines. (b) Two intersecting lines.

non-uniform CAs, demonstrating that high performance systems can be evolved for a number of non-trivial computational tasks. Our results suggest that non-uniformity reduces connectivity requirements, i.e., the use of smaller cellular neighborhoods is made possible.

An important issue when considering systems such as ours is that of scaling, where two separate matters are of concern: the evolutionary algorithm and the evolved solutions. As for the former, namely how does the evolutionary algorithm scale with grid size, we note that as our 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. The second issue is how can larger grids be obtained from smaller (evolved) ones, 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., by [4] for the density task, performance decreases as grid size increases. For non-uniform CAs, quasi-uniformity may facilitate scaling since only a small number of rules must ultimately be considered. To date we have attained successful systems for some tasks using a simple scaling scheme involving the duplication of the rules grid; we are currently exploring a more sophisticated scaling approach, with preliminary encouraging results.

Our work to date suggests that the use of non-uniform CAs coupled with a local, co-evolutionary algorithm offers a number of advantages, including: (1) increased rule variability, thereby entailing easier "adaptation" to a possible change in the "environment", i.e., task, (2) easier implementation as evolware, (3) fault tolerance arising from the insensitivity to minor differences between cellular rules, and (4) better scalability (as noted above).

We found that markedly higher performance is attained for the density task with two-dimensional grids along with shorter computation times, as compared with one-dimensional grids. It is readily observed that a two-dimensional, locally connected grid can be embedded in a one-dimensional grid with local and distant connections. Since the density task is global, it is likely that the observed superior performance of two-dimensional grids arises from the existence of distant connections that enhance information propagation across the grid. This result has motivated the study of a modified model, involving the concomitant evolution of cellular rules and cellular connections. We have found that performance can be markedly increased for global computational tasks by co-evolving connectivity architectures [26, 27].

The nature of computation in CAs is a question of primary importance that has been gaining attention in recent years. We wish to enhance our understanding of the ways CAs perform computations, attempting to gain insight into the laws and mechanisms by which they operate. It is important to learn how CAs may be evolved, rather than designed, to perform computational tasks and what kinds of classes of tasks are most suited for such a computational paradigm. We seek to understand how evolution creates complex, global behavior in locally interconnected systems of simple parts. These goals are significant both from a scientific standpoint as well as from an applicative one.

Evolving, non-uniform CAs hold potential for studying phenomena of interest in areas such as complex systems, artificial life and parallel computation. This work has shed light on the possibility of computing with such CAs, and demonstrated the feasibility of their programming by means of co-evolution. We believe that cellular programming holds potential for attaining evolware which can be implemented in software, hardware, or other possible forms, such as bioware.

Acknowledgments

I am grateful to Daniel Mange, Eytan Ruppin, Eduardo Sanchez, and Marco Tomassini for helpful discussions.

References

1. A. Broggi, V. D'Andrea, and G. Destri. Cellular automata as a computational model for low-level vision. *International Journal of Modern Physics C*, 4(1):5–16, 1993.
2. E. F. Codd. *Cellular Automata.* Academic Press, New York, 1968.

3. J. P. Cohoon, S. U. Hedge, W. N. Martin, and D. Richards. Punctuated equilibria: A parallel genetic algorithm. In J. J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, page 148. Lawrence Erlbaum Associates, 1987.
4. J. P. Crutchfield and M. Mitchell. The evolution of emergent computation. *Proceedings of the National Academy of Sciences USA*, 92(23):10742–10746, 1995.
5. 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.
6. R. Das, M. Mitchell, and J. P. Crutchfield. A genetic algorithm discovers particle-based computation in cellular automata. In Y. Davidor, H. -P. Schwefel, and R. Männer, editors, *Parallel Problem Solving from Nature- PPSN III*, volume 866 of *Lecture Notes in Computer Science*, pages 344–353, Berlin, 1994. Springer-Verlag.
7. P. Gacs. Nonergodic one-dimensional media and reliable computation. *Contemporary Mathematics*, 41:125, 1985.
8. M. Goeke, M. Sipper, D. Mange, A. Stauffer, E. Sanchez, and M. Tomassini. On-line autonomous evolware. In *Proceedings of The First International Conference on Evolvable Systems: from Biology to Hardware (ICES96)*, Lecture Notes in Computer Science. Springer-Verlag, Heidelberg, 1996.
9. Z. Guo and R. W. Hall. Parallel thinning with two-subiteration algorithms. *Communications of the ACM*, 32(3):359–373, March 1989.
10. G. Hernandez and H. J. Herrmann. Cellular-automata for elementary image-enhancement. *CVGIP: Graphical Models and Image Processing*, 58(1):82–89, January 1996.
11. L. Lamport and N. Lynch. Distributed computing: Models and methods. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 1159–1199. Elsevier, 1990.
12. B. Manderick and P. Spiessens. Fine-grained parallel genetic algorithms. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, page 428. Morgan Kaufmann, 1989.
13. M. Mitchell, J. P. Crutchfield, and P. T. Hraber. Dynamics, computation, and the "edge of chaos": A re-examination. In G. Cowan, D. Pines, and D. Melzner, editors, *Complexity: Metaphors, Models and Reality*, pages 491–513. Addison-Wesley, Reading, MA, 1994.
14. M. Mitchell, J. P. Crutchfield, and P. T. Hraber. Evolving cellular automata to perform computations: Mechanisms and impediments. *Physica D*, 75:361–391, 1994.
15. M. Mitchell, P. T. Hraber, and J. P. Crutchfield. Revisiting the edge of chaos: Evolving cellular automata to perform computations. *Complex Systems*, 7:89–130, 1993.
16. N. H. Packard. Adaptation toward the edge of chaos. In J. A. S. Kelso, A. J. Mandell, and M. F. Shlesinger, editors, *Dynamic Patterns in Complex Systems*, pages 293–301. World Scientific, Singapore, 1988.
17. K. Preston, Jr. and M. J. B. Duff. *Modern Cellular Automata: Theory and Applications*. Plenum Press, New York, 1984.
18. S. Rasmussen, C. Knudsen, and R. Feldberg. Dynamics of programmable matter. In C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, editors, *Artificial Life II*, volume X of *SFI Studies in the Sciences of Complexity*, pages 211–254, Redwood City, CA, 1992. Addison-Wesley.

19. E. Sanchez and M. Tomassini, editors. *Towards Evolvable Hardware*, volume 1062 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1996.
20. M. Sipper. Non-uniform cellular automata: Evolution in rule space and formation of complex structures. In R. A. Brooks and P. Maes, editors, *Artificial Life IV*, pages 394–399, Cambridge, Massachusetts, 1994. The MIT Press.
21. M. Sipper. An introduction to artificial life. *Explorations in Artificial Life (special issue of AI Expert)*, pages 4–8, September 1995. Miller Freeman, San Francisco, CA.
22. M. Sipper. Quasi-uniform computation-universal cellular automata. In F. Morán, A. Moreno, J. J. Merelo, and P. Chacón, editors, *ECAL'95: Third European Conference on Artificial Life*, volume 929 of *Lecture Notes in Computer Science*, pages 544–554, Berlin, 1995. Springer-Verlag.
23. M. Sipper. Studying artificial life using a simple, general cellular model. *Artificial Life Journal*, 2(1):1–35, 1995. The MIT Press, Cambridge, MA.
24. M. Sipper. Co-evolving non-uniform cellular automata to perform computations. *Physica D*, 92:193–208, 1996.
25. M. Sipper. Complex computation in non-uniform cellular automata, 1996. (submitted).
26. M. Sipper and E. Ruppin. Co-evolving architectures for cellular machines. *Physica D*, 1996. (to appear).
27. M. Sipper and E. Ruppin. Co-evolving cellular architectures by cellular programming. In *Proceedings of IEEE Third International Conference on Evolutionary Computation (ICEC'96)*, pages 306–311, 1996.
28. T. Starkweather, D. Whitley, and K. Mathias. Optimization using distributed genetic algorithms. In H. -P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature*, volume 496 of *Lecture Notes in Computer Science*, page 176, Berlin, 1991. Springer-Verlag.
29. R. Tanese. Parallel genetic algorithms for a hypercube. In J. J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, page 177. Lawrence Erlbaum Associates, 1987.
30. T. Toffoli and N. Margolus. *Cellular Automata Machines*. The MIT Press, Cambridge, Massachusetts, 1987.
31. M. Tomassini. The parallel genetic cellular automata: Application to global function optimization. In R. F. Albrecht, C. R. Reeves, and N. C. Steele, editors, *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms*, pages 385–391. Springer-Verlag, 1993.
32. M. Tomassini. A survey of genetic algorithms. In D. Stauffer, editor, *Annual Reviews of Computational Physics*, volume III, pages 87–118. World Scientific, 1995. Also available as: Technical Report 95/137, Department of Computer Science, Swiss Federal Institute of Technology, Lausanne, Switzerland, July, 1995.
33. S. Wolfram. Statistical mechanics of cellular automata. *Reviews of Modern Physics*, 55(3):601–644, July 1983.
34. S. Wolfram. Universality and complexity in cellular automata. *Physica D*, 10:1–35, 1984.