



ELSEVIER

BioSystems 42 (1997) 29–43



The evolution of parallel cellular machines: Toward evolware

Moshe Sipper *

Logic Systems Laboratory, Swiss Federal Institute of Technology, IN-Ecublens, CH-1015 Lausanne, Switzerland

Received 24 June 1996; accepted 1 November 1996

Abstract

The historical idea of evolving machines has recently resurfaced as the nascent field of bio-inspired systems and evolvable hardware. This paper describes the cellular programming approach used to evolve parallel cellular machines, presenting its application to six computational problems: density, synchronization, ordering, boundary computation, thinning and random number generation. Our results show that successful machines can be evolved to solve these tasks. The methodology described herein represents one possible approach to attaining truly evolving ware, evolware, with current implementations centering on hardware, while raising the possibility of using other forms in the future, such as bioware. The paper presents work in progress, the aim being to give an account of results obtained to date, ending with a list of several open issues for future research. © 1997 Elsevier Science Ireland Ltd.

Keywords: Non-uniform cellular automata; Cellular programming; Evolutionary computation; Complex adaptive systems; Evolware

1. Introduction

The idea of applying the biological principle of natural evolution to artificial systems, introduced more than three decades ago, has seen an impressive growth in the past few years. Usually grouped under the term evolutionary algorithms or evolutionary computation, one finds such diverse domains as genetic algorithms, evolution strategies, evolutionary programming, and genetic programming. Central to all these different

methodologies is the idea of solving problems by evolving an initially random pool of possible solutions, through the application of ‘genetic’ operators, such that in time ‘fitter’ (i.e., better) solutions emerge (Bäck, 1996; Michalewicz, 1996; Mitchell, 1996; Fogel, 1995; Goldberg, 1989; Holland, 1975).

Research in these areas has traditionally centered on proving theoretical aspects, such as convergence properties, effects of different algorithmic parameters, and so on, or on making headway in new application domains, such as constraint optimization problems, image process-

* E-mail: Moshe.Sipper@di.epfl.ch

ing, neural network evolution and more. The implementation of an evolutionary algorithm, an issue which usually remains in the background, is quite costly in many cases, since populations of solutions are involved, possibly coupled with computation-intensive fitness evaluations. One possible solution is to parallelize the process, an idea which has been explored to some extent in recent years (Tomassini, 1996; Cantú-Paz, 1995). While posing no major problems in principle, this may require judicious modifications of existing algorithms or the introduction of new ones in order to meet the constraints of a given parallel machine.

In this paper a different approach is taken; rather than ask ourselves how to implement better a specific algorithm on a given hardware platform, we pose the more general question of whether machines can be made to evolve. While this idea finds its origins in the cybernetics movement of the 1940s and 1950s, it has recently resurged in the form of the nascent field of bio-inspired systems and evolvable hardware (Sanchez and Tomassini, 1996). The field draws on ideas from evolutionary computation as well as on recent hardware developments.

This study describes research into evolving cellular machines. We introduce the basic approach, denoted as cellular programming, and demonstrate its viability by studying a number of non-trivial computational problems. Work in progress is presented, the aim being to give an account of results obtained to date. Several questions are as yet left unanswered, hopefully to be addressed in future work; we have attempted to assemble these in the final section. Though the results described have been obtained through software simulation, the ultimate goal is to attain truly ‘evolving ware’—evolware—with current implementations centering on hardware, while raising the possibility of using other forms in the future, such as bioware. We have recently implemented an evolving, on-line, autonomous hardware system based on the cellular programming approach described in this paper (Goeke et al., 1996).

Our evolving machines are based on the cellular automata model. 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 (Wolfram, 1984; Toffoli and Margolus, 1987). The cellular array (grid) is n -dimensional, where $n = 1, 2, 3$ is used in practice (in this work we shall concentrate on $n = 1$ and $n = 2$). A one-dimensional CA is illustrated in Fig. 1 (Mitchell, 1996).

CAs exhibit three notable features, namely massive parallelism, locality of cellular interactions, and simplicity of basic components (cells). As such they are naturally suited to hardware implementation, with the potential of exhibiting extremely fast and reliable computation that is robust to noisy input data and component failure (Gacs, 1985). 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 to perform a particular task is highly complicated, thus severely limiting their applications. This results from the local dynamics of the system, which renders the design

Rule table:

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

Grid:

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

Fig. 1. Illustration of a one-dimensional, 2-state CA. The connectivity radius is $r = 1$, meaning that each cell has two neighbours, one to its immediate left and one to its immediate right. Grid size is $N = 15$. The rule table for updating the grid is shown on top. The grid configuration over one time step is shown at the bottom. Spatially periodic boundary conditions are applied, meaning that the grid is viewed as a circle, with the leftmost and rightmost cells each acting as the other’s neighbour.

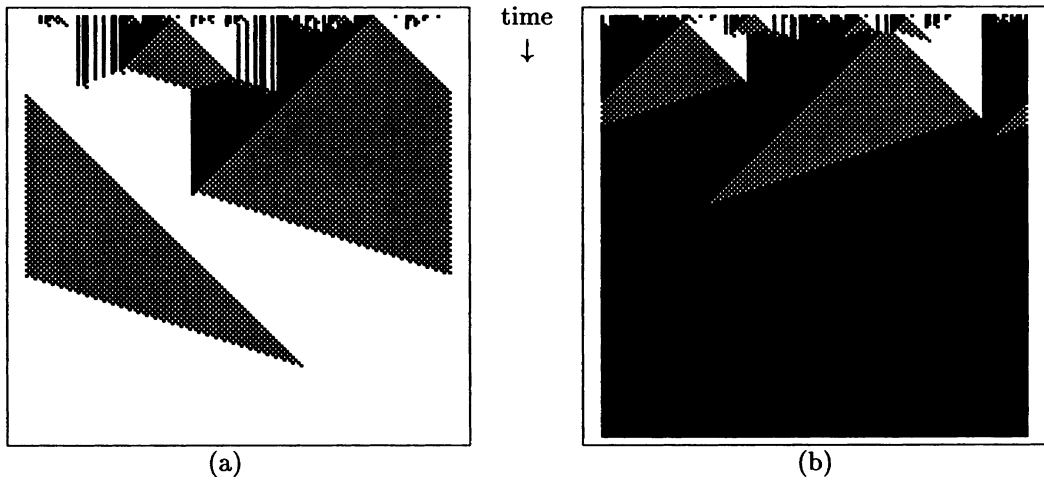


Fig. 2. The density task: operation of the GKL rule. CA is one-dimensional, uniform, 2-state, with connectivity radius $r = 3$. Grid size is $N = 149$. White squares represent cells in state 0, black squares represent cells in state 1. The pattern of configurations is shown through time (which increases down the page). Initial configurations were generated at random. (a) Initial density of 1s is 0.47. (b) Initial density of 1s is 0.53. The CA relaxes in both cases to a fixed pattern of all 0s or all 1s, correctly classifying the initial configuration.

of local rules to perform global computational tasks extremely arduous. Automating the design (programming) process would greatly enhance the viability of CAs (Mitchell et al., 1994b).

The model investigated in this paper is an extension of the CA model, termed non-uniform cellular automata (Sipper, 1994; Vichniac et al., 1986). 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, using the cellular programming approach.

Section 2 describes previous work on non-uniform CAs and evolving CAs. The cellular programming algorithm is delineated in Section 3, and applied to six computational tasks in Section 4: density, synchronization, ordering, boundary computation, thinning, and random number generation. We demonstrate that parallel cellular machines can be successfully evolved to solve these tasks. Our findings and future work are discussed in Section 5.

2. Previous work

The application of genetic algorithms to the evolution of uniform cellular automata was initially studied by Packard (1988) and recently undertaken by the EVCA (evolving CA) group (Mitchell et al., 1993, 1994a,b; Das et al., 1994, 1995; Crutchfield and Mitchell, 1995). They carried out experiments involving 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). (For two-dimensional CAs, two types of cellular neighborhoods are usually considered: 5-neighbor, consisting of the cell along with its four immediate nondiagonal neighbors, and 9-neighbor, consisting of the cell along with its eight surrounding neighbors.) 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 (Fig. 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 (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 next-state (output) bits for all possible neighborhood configurations, listed in lexicographic order (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, in accordance with the task at hand. Using the genetic algorithm highly successful CA rules were found for both the density and the synchronization tasks.

The model investigated in this paper is that of non-uniform CAs, where cellular rules need not be identical for all cells. Thus, rather than seek a single rule that must be universally applied to all cells in the grid, each cell is allowed to ‘choose’ its own rule through evolution. As we shall see, the removal of the uniformity constraint from the original CA model lends itself to a novel algorithm which is more amenable to implementation as *evolware*, in comparison to standard evolutionary algorithms.

We have previously applied the non-uniform CA model to the investigation of artificial life issues (Sipper, 1994; 1995c,a), and have also demonstrated its computation-universality (Sipper, 1995b). The co-evolution of non-uniform, one-dimensional CAs to perform computations was first undertaken in Sipper (1996a). We presented results pertaining to the density task, showing that high performance, non-uniform CAs can be co-evolved not only with radius $r=3$ (Mitchell et al., 1994b), but also for smaller radii, most notably $r=1$ which is minimal. Sipper (1996b) and Goeke et al. (1996) 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, 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 (Sipper, 1995b, 1996a).

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 (the ‘genome’), containing the next-state (output) bits for all possible neighborhood configurations (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, and for each one 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 the algorithm is delineated in Fig. 3.

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/bit.

There are two main differences between the cellular programming algorithm and the standard genetic algorithm: (a) The latter involves a population of evolving, uniform CAs; all CAs are ranked according to fitness, with crossover occur-

```

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  $c \bmod C = 0$  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
        (this case can occur if the cellular neighborhood includes
        more than two cells)
    end if
     $f_i = 0$ 
  end parallel for
  end if
end while

```

Fig. 3. Pseudo-code of the cellular programming algorithm.

ring 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, the cellular programming 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 may also be considered a form of symbiotic cooperation, which falls, as does co-evolution, under the general heading of 'ecological' interactions (Mitchell, 1996 pages 182–183).)

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, thereby decreasing computation time and enhancing performance (Tomassini, 1995). A number of models

Table 1
List of computational tasks for which cellular machines were evolved via cellular programming

Task	Description	Grid
Density	Decide whether the initial configuration contains a majority of 0s or of 1s	1D, $r = 1$ 2D, 5-neighbor
Synchronization	Given any initial configuration, relax to an oscillation between all 0s and all 1s	1D, $r = 1$ 2D, 5-neighbor
Ordering	Order initial configuration so that 0s are placed on the left and 1s are placed on the right	1D, $r = 1$
Rectangle-boundary	Find the boundaries of a randomly-placed, random-sized non-filled rectangle	2D, 5-neighbor
Thinning	Find thin representations of rectangular patterns	2D, 5-neighbor
Random number	Generate 'good' sequences of pseudo-random numbers	1D, $r = 1$

have been suggested, among them coarse-grained, island models (Starkweather et al., 1991; Cohoon et al., 1987; Tanese, 1987), and fine-grained, grid models (Tomassini, 1993; Manderick and Spiessens, 1989). 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 six 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), thinning (Section 4.5) and random number generation (Section 4.6); these are summarized in Table 1. Minimal cellular spaces are used: two-state, $r = 1$ for the one-dimensional case and two-state, five-neighbor for the two-dimensional one. Spatially periodic boundary conditions are applied, resulting in a circular grid for the one-dimensional case, and a toroidal one for the two-dimensional case. 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 configura-

tions, 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 Mitchell et al. (1994b), the density task comprises a non-trivial computation for a small radius CA ($r \ll N$, where N is the grid size). 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. Note that the density task cannot be perfectly solved by a uniform, two-state CA, as proven by Land and Belew (1995a). (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 (Capcarrere et al., 1996).)

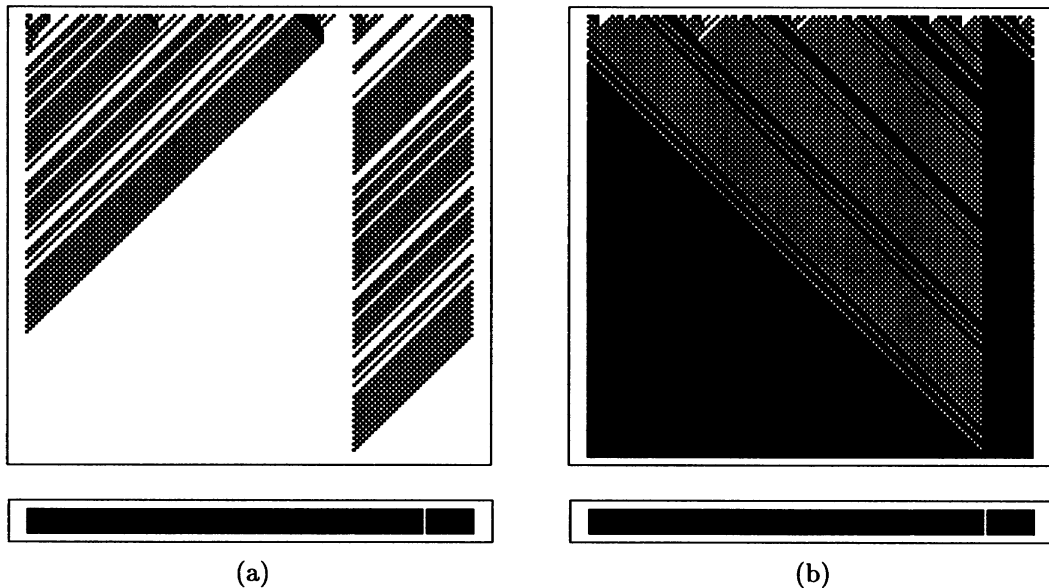


Fig. 4. One-dimensional density task: operation of a co-evolved, non-uniform, $r = 1$ CA. Grid size is $N = 149$. Top figures depict space-time diagrams, bottom figures depict rule maps. (a), Initial density of 1s is 0.40, final density is 0. (b), Initial density of 1s is 0.60, final density is 1.

The operation of the human-designed GKL rule, which exhibits high performance on this task, is shown in Fig. 2 (Sipper, 1996a; Mitchell et al., 1994b; Gonzaga de Sá and Maes, 1992; Gacs et al., 1978). We observe that propagation of information about local neighborhoods takes place to produce the final fixed-point configuration. Essentially, the rule's 'strategy' is to successively classify local densities, with the locality range increasing over time. In regions of ambiguity a 'signal' is propagated, seen either as a checkerboard pattern in space-time or as a vertical white-to-black boundary (a detailed analysis of this rule is provided in Mitchell et al. (1994b) and Crutchfield and Mitchell (1995)).

We have studied this task (Sipper, 1996a; Sipper and Ruppin, 1996a,b) 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 on the density task, a feat not possible for larger

values of r . One of our major results is that evolved non-uniform, $r = 1$ CAs out perform any possible uniform, $r = 1$ CA (Sipper, 1996a). For details on the performance comparison see (Sipper, 1996a).

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 performance values as high as 0.93. (In comparison, the maximal performance of uniform $r = 1$ CAs is 0.83 (Sipper, 1996a)). Furthermore, these consist of a grid in which one rule dominates, a situation referred to as quasi-uniformity (Section 2). Fig. 4 demonstrates the operation of one such co-evolved CA along with a rules map, depicting the distribution of rules by assigning a unique gray level to each distinct rule. In this example the grid consists of 146 cells containing rule 226, two cells containing rule 224, and one cell containing rule 234. Rule numbers are given in accordance with Wolfram's convention (Wolfram, 1983), representing the decimal equiva-

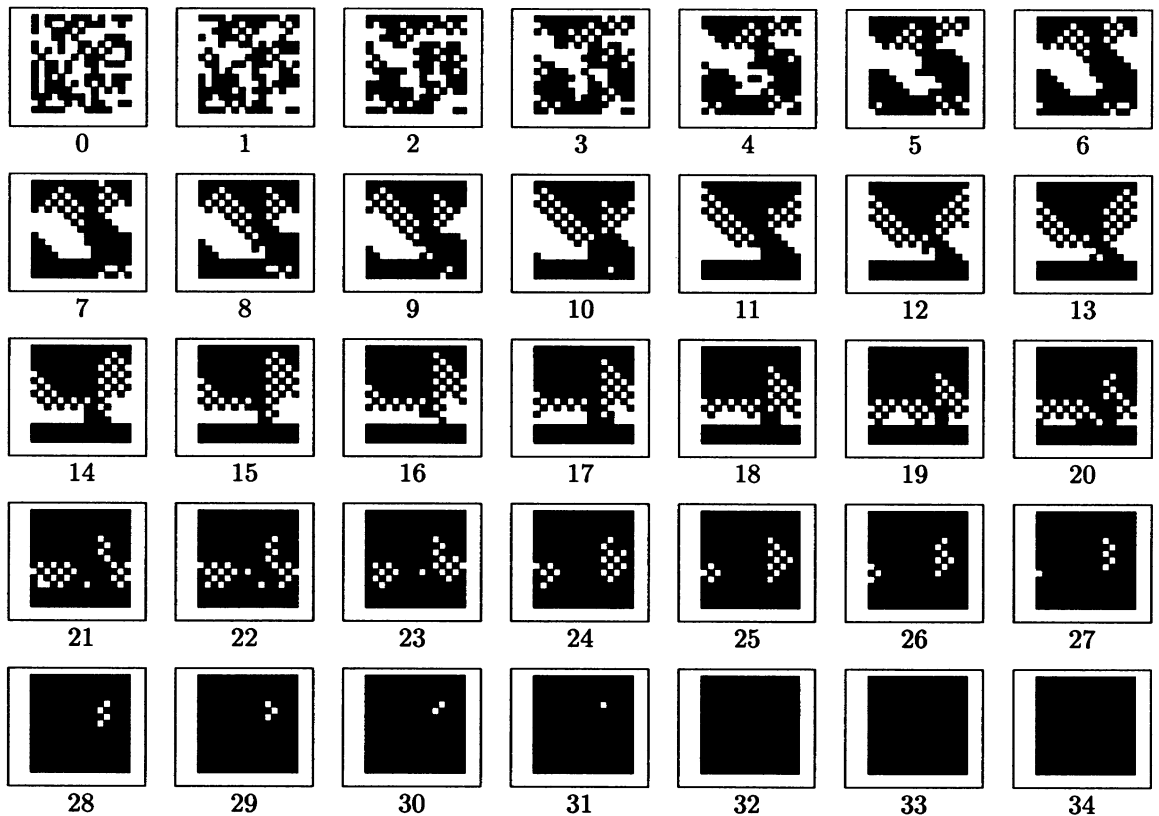


Fig. 5. Two-dimensional density task: operation of a co-evolved, non-uniform, 2-state, 5-neighbour CA. Grid size is $N = 225$ (15×15). Initial density of 1s is 0.51, final density is 1. Numbers at bottom of images denote time steps.

lent of the binary number encoding the rule table. For example, the rule depicted in Fig. 1 is rule 232. The non-dominant rules act as ‘buffers’ preventing information from flowing too freely, and making local corrections to passing signals. A detailed investigation of the application of cellular programming to the one-dimensional density task can be found in Sipper (1996a) and Sipper and Ruppin (1996a).

The density task can be extended in a straightforward manner to two-dimensional grids, an investigation of which we have carried out, attaining a notably higher performance than the one-dimensional case, with values of 0.99. Fig. 5 demonstrates the operation of one such co-evolved CA. Qualitatively, we observe the CA’s ‘strategy’ of successively classifying local densities, with the locality range increasing over time; ‘com-

peting’ 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 Das et al. (1995) and studied by us in Sipper (1996b), Goeke et al. (1996) and Sipper (1997) 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. As with the density task, synchronization also comprises a non-trivial computation for a small-radius CA.

We studied non-uniform, one-dimensional, minimal radius $r = 1$ CAs of size $N = 149$; as for

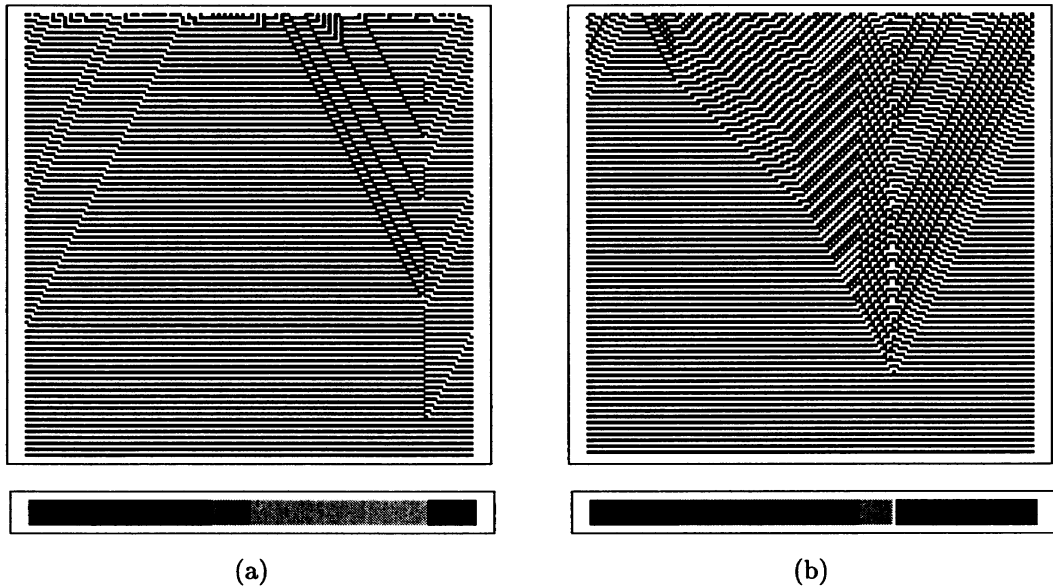


Fig. 6. One-dimensional synchronization task: Operation of two-co-evolved, non-uniform, $r = 1$ CAs. Grid size is $N = 149$. Top figures depict space-time diagrams, bottom figures depict rule maps.

the density task, all possible uniform, $r = 1$ CAs were first tested on this task. 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. We found that quasi-uniform CAs had co-evolved that exhibit near-perfect performance, which surpasses any possible uniform, $r = 1$ CA. Fig. 6 depicts the operation of two such co-evolved CAs, along with rule maps. We have also experimented with two-dimensional grids, obtaining highly successful results as with the one-dimensional case.

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 (thus the final density equals the initial one, however the configuration consists of a block of 0s on the left followed by a block of 1s on the right). 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. Cellular programming

yielded quasi-uniform CAs with fitness values as high as 0.93, one of which is depicted in Fig. 7. As with the previous tasks we were able to ascertain that this performance level is better than any possible uniform $r = 1$ 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 by Preston and Duff (1984), with more recent applications presented in Broggi et al. (1993) and Hernandez and Herrmann (1996).

The next two tasks 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

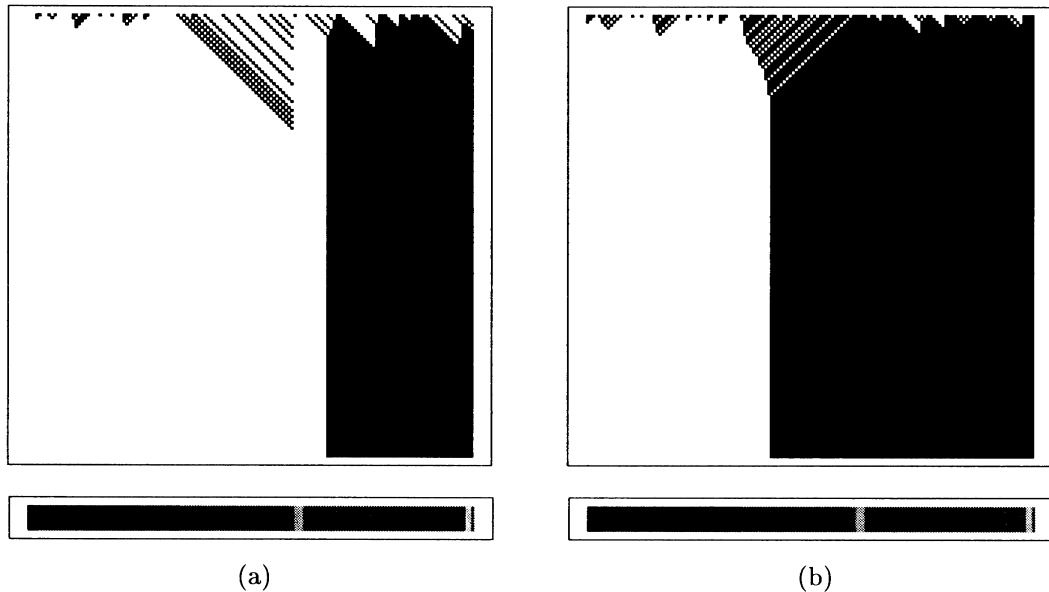


Fig. 7. One-dimensional ordering task: operation of a co-evolved, non-uniform, $r = 1$ CA. Top figures depict space–time diagrams. Bottom figures depict rule maps. (a), Initial density of 1s is 0.315, final density is 0.328. (b), Initial density of 1s is 0.60, final density is 0.59.

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 be considered a form of image enhancement, e.g. used for treating corrupted images. Using cellular programming, non-uniform CAs were evolved with performance values of 0.99, one of which is depicted in Fig. 8.

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 (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 (Wolfram, 1983)). 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 stroke or curve. Such 'thinned' representations are typically

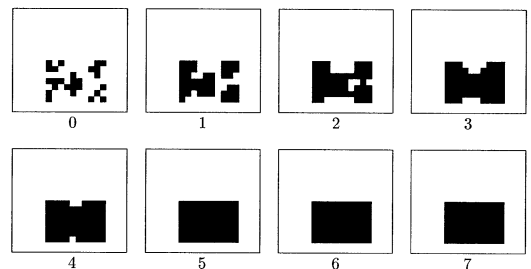


Fig. 8. Two-dimensional rectangle-boundary task: operation of a co-evolved, non-uniform, 2-state, 5-neighbour CA. Grid size is $N = 225$ (15×15). Numbers at bottom of images denote time steps.

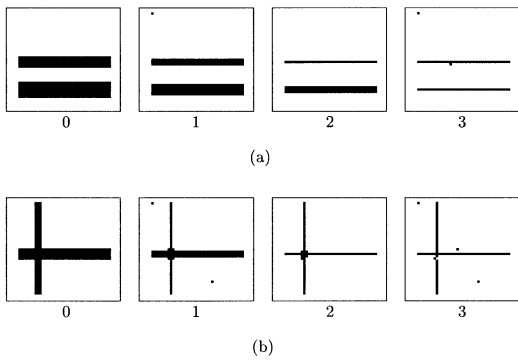


Fig. 9. Two-dimensional thinning task: operation of a co-evolved, non-uniform, 2-state, 5-neighbour CA. Grid size is $N = 1600$ (40×40). Numbers at bottom of images denote time steps. (a), Two separate lines. (b), Two intersecting lines.

easier to process in later stages, entailing savings in both time and storage space (Guo and Hall, 1989).

While the first thinning algorithms were designed for serial implementation, current interest lies in parallel systems, early examples of which were presented in Preston and Duff (1984). 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. Guo and Hall (1989) considered four sets of binary images, 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 consisting 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 Guo and Hall (1989), it is noted that we employ a smaller neighborhood (5-cell) and do not apply any sub-iterations.

Fig. 9 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 (Fig. 9a). Furthermore, partial success is demonstrated when presented

with more difficult images involving intersecting lines (Fig. 9b).

4.6. Random number generation

Random numbers are needed in a variety of applications, yet finding good random number generators, or randomizers, is a difficult task (Park and Miller, 1988). To generate a random sequence on a digital computer, one starts with a fixed length seed, then iteratively applies some transformation to it, progressively extracting as long as possible a random sequence. Such numbers are usually referred to as pseudo-random, as distinguished from true random numbers resulting from some natural physical process. In the last decade cellular automata have been used to generate random numbers (Wolfram, 1986; Hortensius et al., 1989; Koza, 1992).

Sipper and Tomassini (1996a,b) applied the cellular programming algorithm to evolve random number generators. Essentially, the cell's fitness score for a single configuration (Fig. 3) is the entropy of the temporal bit sequence of that cell, with higher entropy implying better fitness. This fitness measure was used to drive the evolutionary process, after which standard tests were applied to evaluate the quality of the evolved CAs. The results obtained suggest that good generators can indeed be evolved; these exhibit behavior at least as good as that of previously described CAs, with notable advantages arising from the existence of a 'tunable' algorithm for obtaining random number generators.

5. Concluding remarks and future work

In this paper we described the cellular programming approach used to evolve parallel cellular machines, and demonstrated its viability by applying it to the solution of six computational problems. This methodology represents one possible approach to attaining truly evolving ware, evolware. Indeed, we have recently implemented an evolving, on-line, autonomous hardware system based on cellular programming (Goeke et al., 1996).

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 (see below); and (4) better scalability (see below).

The work reported herein represents a first step in an exciting, nascent domain. While results to date are encouraging, there are still several possible avenues for future research, some of which have been explored to a certain extent, while others await to be pursued. We have attempted to assemble some of these below.

What classes of computational tasks are most suitable for such evolving cellular machines? and, what possible applications do they entail? We have noted feasible application areas such as image processing and random number generation. Clearly, more research is necessary in order to elaborate these directions as well as to find new ones.

Computation in cellular machines—how are we to understand the emergent, global computation arising in such locally-connected machines? Crutchfield and Mitchell (1995) and Das et al. (1994, 1995) carried out an interesting analysis using automated methods developed by Crutchfield and Young (1989), Hanson and Crutchfield (1992) and Crutchfield and Hanson (1993) for discovering computational structures embedded in the space–time behavior of CAs. Currently, we have performed a more in-depth analysis within the context of the cellular programming framework in Sipper (1996a). This issue is interesting both from a theoretical point of view as well as from a practical one, where it may help guide our search for suitable classes of tasks for such machines.

Studying the evolutionary process—Mitchell et al. (1994b, 1993) investigated the process of evolution (of uniform CAs) embodied by the standard genetic algorithm. Our algorithm is different, as it is local and co-evolutionary, presenting novel and interesting dynamics worthy of further study. We

wish to enhance our understanding of how evolution creates complex, global behavior in such locally interconnected systems of simple parts; a first step along this path has been taken in Sipper (1996a).

Modifications of the evolutionary algorithm—the representation of CA rules (i.e. the ‘genome’) used in the experiments reported herein consists of a bit string containing a lexicographic listing of all possible neighborhood configurations (Section 2). It has been noted by Land and Belew (1995b) that this representation is fairly low level and brittle since a change of one bit in the rule table can have a large effect on the computation performed. They evolved uniform CAs to perform the density task using other bit-string representations, as well as a novel, higher-level one consisting of condition–action pairs; it was demonstrated that better performance is attained when employing the latter. More recently, Andre et al. (1996a,b) used genetic programming (Koza, 1992), in which a rule is represented by a LISP expression, to evolve uniform CAs to perform the density task; this resulted in a CA which outperforms the best known uniform, $r=3$ density classifier, namely the GKL rule. These experiments demonstrate that changing the bit-string representation, i.e. the encoding of the ‘genome’ may entail notable performance gains; indeed, this issue is of prime import where evolutionary algorithms in general are concerned (Mitchell, 1996). Such encodings could be incorporated into the cellular programming approach. We noted in Section 3 that fitness is assigned locally to each cell; another possibility might be to assign fitness scores to blocks of cells, in accordance with their mutual degree of success on the task at hand. It is clear that the novelty of our algorithm leaves much to be explored, and we plan to incorporate such ideas, as well as others, within our framework.

Modifications of the cellular machine model—in this paper we studied one generalization of the CA model involving non-uniformity of rules. Other possible modifications include: (1) Generalizing upon the CA’s standard, homogeneous connectivity. We have studied non-standard connectivity architectures, where each cell has a

small, identical number of connections, yet not necessarily from its most immediate neighboring cells, and also investigated the possibility of evolving the architecture through a two-level evolutionary process, in which the cellular rules evolve concomitantly with the cellular connections. It was shown that definite performance gains can thus be attained (Sipper and Ruppin, 1996a,b); (2) The application of asynchronous state updating (currently synchronous updating is applied, i.e. all cells are updated at once) (Sipper et al., 1996b) (this issue was previously investigated by (Sipper, 1995c) in a somewhat different model); (3) Non-deterministic updating, also connected with the issue of robustness, namely how resistant are our systems in the face of errors (e.g. how is the computation affected when a small probability of error is introduced in the updating of cell states?) (Sipper et al., 1996a,b); and (4) Three-dimensional grids (and tasks).

One of the motivations for the above modifications of the cellular machine model is the desire to attain realistic systems that are more amenable to implementation as *evolware*.

Scaling—this involves two separate issues: the evolutionary algorithm and the evolved solutions.

How does the evolutionary algorithm scale with grid size? Though to date we have performed experiments with different grid sizes, a more in depth inquiry is needed. 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.

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 Crutchfield and Mitchell (1995) 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 the random number generation task using a simple scaling scheme involving the duplication of the rules grid (Sipper and Tomassini, 1996a). We are currently exploring a more sophisticated scaling approach, with preliminary encouraging results.

Implementation—as discussed above, this is one of the prime motivations of our work, the goal being to construct *evolware*.

Evolving cellular machines hold potential both scientifically, as vehicles for studying phenomena of interest in areas such as complex systems and artificial life, as well as practically, showing a range of potential future applications ensuing from the construction of adaptive systems. This work has shed light on the possibility of computing with such machines, and demonstrated the feasibility of their programming by means of co-evolution.

Acknowledgements

I wish to thank David Fogel, Daniel Mange, Melanie Mitchell, Eytan Ruppin, and Eduardo Sanchez for their valuable comments and suggestions. I am especially grateful to Marco Tomassini for our many stimulating discussions, the results of which are disseminated throughout this paper.

References

- Andre, D., Bennett III, F.H. and Koza, J.R. 1996a, Discovery by genetic programming of a cellular automata rule that is better than any known rule for the majority classification problem, in: *Genetic Programming 1996: Proc. 1st Ann. Conf.*, J.R. Koza, D.E. Goldberg, D.B. Fogel and R.L. Riolo (eds.) (MIT Press, Cambridge, MA) pp. 3–11.
- Andre, D., Bennett III, F.H. and Koza J.R. 1996b, Evolution of intricate long-distance communication signals in cellular automata using genetic programming, in: *Artificial Life V: Proc. 5th Int. Workshop on the Synthesis and Simulation of Living Systems*, C. Langton and T. Shimohara (eds.) (MIT Press, Cambridge, MA).
- Bäck, T., 1996, *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms* (Oxford University Press, New York).
- Broggi, A., D'Andrea, V. and Destri, G., 1993, Cellular automata as a computational model for low level vision. *Int. J. Mod. Phy.* 4 (1), 5–16.

- Cantú-Paz, E., 1995, A summary of research on parallel genetic algorithms. Technical Report 95007, Illinois Genetic Algorithms Laboratory (University of Illinois at Urbana-Champaign, Urbana, IL).
- Capcarrere, M. S., Sipper, M. and Tomassini M. 1996, A two-state, $r=1$ cellular automaton that classifies density. *Phys. Rev. Lett.* 77, 4969–4979.
- Cohon, J.P., Hedge, S.U., Martin, W.N. and Richards, D., 1987, Punctuated equilibria: a parallel genetic algorithm, in: *Proc. 2nd Int. Conf. on Genetic Algorithms*, J.J. Grefenstette, (ed.) (Lawrence Erlbaum Associates) p. 148.
- Crutchfield, J.P. and Hanson, J.E., 1993, Turbulent pattern bases for cellular automata. *Physica D* 69, 279–301.
- Crutchfield, J.P. and Mitchell, M., 1995, The evolution of emergent computation. *Proc. Natl. Acad. Sci. USA* 92 (23), 10 742–10 746.
- Crutchfield, J.P. and Young, K., 1989, Conferring statistical complexity. *Phys. Rev. Lett.* 63, 105.
- Das, R., Mitchell, M. and Crutchfield, J.P., 1994, A genetic algorithm discovers particle-based computation in cellular automata. in: *Parallel Problem Solving from Nature-PPSN III, Lecture Notes in Computer Science*, Vol. 866, Y. Davidor, H.-P. Schwefel and R. Männer (eds.) (Springer-Verlag, Berlin) pp. 344–353.
- Das, R., Crutchfield, J.P., Mitchell, M. and Hanson, J.E., 1995, Evolving globally synchronized cellular automata, in: *Proc. 6th Int. Conf. on Genetic Algorithms*, L.J. Eshelman (ed.) (Morgan Kaufmann, San Francisco, CA) pp. 336–343.
- Fogel, D.B., 1995, *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence* (IEEE Press, Piscataway, NJ).
- Gacs, P., Kurdyumov, G.L. and Levin, L.A., 1978, One-dimensional uniform arrays that wash out finite islands. *Probl. Peredachi Inf.* 14, 92–98.
- Gacs, P., 1985, Nonergodic one-dimensional media and reliable computation. *Contemp. Math.* 41, 125.
- Goeke, M., Sipper, M., Mange, D., Stauffer, A., Sanchez, E. and Tomassini, M., 1996, Online autonomous evolware, in: *Proc. 1st Int. Conf. on Evolvable Systems: from Biology to Hardware (ICES96)*, Lecture Notes in Computer Science (Springer-Verlag, Heidelberg) (in press).
- Goldberg, D.E., 1989, *Genetic Algorithms in Search, Optimization and Machine Learning* (Addison-Wesley, Reading, MA).
- Gonzaga de Sá, P. and Maes, C., 1992, The Gacs-Kurdyumov-Levin automaton revisited. *J. Stat. Phys.* 67 (3/4), 507-522.
- Guo, Z. and Hall, R.W., 1989, Parallel thinning with two-subiteration algorithms. *Comm. ACM* 32 (3), 359–373.
- Hanson, J.E. and Crutchfield, J.P., 1992, The attractor-basin portrait of a cellular automaton. *J. Stat. Phys.* 66, 1415–1462.
- Hernandez, G. and Herrmann, H.J., 1996, Cellular-automata for elementary image-enhancement. *CVGIP: Graph. Models Image Process.* 58 (1), 82–89.
- Holland, J.H., 1975, *Adaptation in Natural and Artificial Systems* (The University of Michigan Press, Ann Arbor, MI).
- Hortensius, P.D., McLeod, R.D. and Card, H.C., 1989, Parallel random number generation for VLSI systems using cellular automata. *IEEE Trans. Comput.* 38 (10), 1466–1473.
- Koza, J.R., 1992, *Genetic Programming*. (MIT Press, Cambridge, MA).
- Land, M. and Belew, R.K., 1995a, No perfect two-state cellular automata for density classification exists. *Phys. Rev. Lett.* 74 (25), 5148–5150.
- Land, M. and Belew, R.K., 1995b Towards a self-replicating language for computation, in: *Evolutionary programming IV: Proc. 4th Ann. Conf. on Evolutionary Programming*, J.R. McDonnell, R.G. Reynolds and D.B. Fogel (eds.) (MIT Press, Cambridge, MA) pp. 403–413.
- Manderick, B. and Spiessens, B., 1989, Fine-grained parallel genetic algorithms, in: *Proc. 3rd Int. Conf. on Genetic Algorithms*, J.D. Schaffer (ed.) (Morgan Kaufmann) p. 428.
- Michalewicz, Z., 1996, *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd edn. (Springer-Verlag, Berlin).
- Mitchell, M., Hraber, P.T. and Crutchfield, J.P., 1993, Revisiting the edge of chaos: evolving cellular automata to perform computations. *Complex Syst.* 7, 89-130.
- Mitchell, M., Crutchfield, J.P. and Hraber, P.T., 1994a, Dynamics, computation, and the ‘edge of chaos’: a re-examination, in: *Complexity: Metaphors, Models and Reality*, G. Cowan, D. Pines and D. Melzner (eds.) (Addison-Wesley, Reading, MA) pp. 491–513.
- Mitchell, M., Crutchfield, J.P. and Hraber, P.T., 1994b, Evolving cellular automata to perform computations: mechanisms and impediments. *Physica D* 75, 361–391.
- Mitchell, M., 1996, *An Introduction to Genetic Algorithms* (MIT Press, Cambridge, MA).
- Packard, N.H., 1988, Adaptation toward the edge of chaos, in: *Dynamic Patterns in Complex Systems*, J.A.S. Kelso, A.J. Mandell and M.F. Shlesinger (eds.) (World Scientific, Singapore) pp. 293–301.
- Park, S.K. and Miller, K.W., 1988, Random number generators: good ones are hard to find. *Commun. ACM*, 31 (10), 1192–1201.
- Preston, Jr. K. and Duff, M.J.B., 1984, *Modern Cellular Automata: Theory and Applications*. (Plenum, New York).
- Sanchez, E. and Tomassini, M. (eds.), 1996, *Towards Evolvable Hardware*, Lecture Notes in Computer Science, Vol. 1062 (Springer-Verlag, Berlin).
- Sipper, M. and Ruppig, E., 1996a, Co-evolving architectures for cellular machines. *Physica D* 99, 428–441.
- Sipper, M. and Ruppig, E., 1996b, Co-evolving cellular architectures by cellular programming, in: *Proc. IEEE 3rd Int. Conf. on Evolutionary Computation (ICEC’96)*, pp. 306–311.
- Sipper, M. and Tomassini, M., 1996a, Co-evolving parallel random number generators, in: *Parallel Problem Solving*

- from Nature—PPSN IV, Lecture Notes in Computer Science, Vol. 1141, H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel (eds.) (Springer-Verlag, Heidelberg) pp. 950–959.
- Sipper, M. and Tomassini, M., 1996b, Generating parallel random number generators by cellular programming. *Int. J. Mod. Phys. C* 7 (2), 181–190.
- Sipper, M., Tomassini, M. and Beuret, O., 1996a Studying probabilistic faults in evolved non-uniform cellular automata. *Int. J. Mod. Phys. C* 7, 923–939.
- Sipper, M., Tomassini, M. and Capcarrere, M., 1996b, Designing cellular automata using a parallel evolutionary algorithm. *Nucl. Instr. Methods Phys. Res. A* (in press).
- Sipper, M., 1994, Non-uniform cellular automata: evolution in rule space and formation of complex structures, in: *Artificial Life IV*, R.A. Brooks and P. Maes (eds) (MIT Press, Cambridge, MA) pp. 394–399.
- Sipper, M., 1995a, An introduction to artificial life, in: *Explorations in Artificial Life* (special issue *AI Expert*), (Miller Freeman, San Francisco, CA) pp. 4–8.
- Sipper, M., 1995b, Quasi-uniform computation-universal cellular automata, in: *ECAL'95: 3rd Eur. Conf. on Artificial Life*, Lecture Notes in Computer Science, Vol. 929, F. Morán, A. Moreno, J.J. Merelo and P. Chacón (eds.) (Springer-Verlag, Berlin) pp. 544–554.
- Sipper, M., 1995c, Studying artificial life using a simple, general cellular model. *Artif. Life J.* 2 (1), 1–35.
- Sipper, M., 1996a, Co-evolving non-uniform cellular automata to perform computations. *Physica D* 92, 193–208.
- Sipper, M., 1996b, Designing evolware by cellular programming, in: *Proc. 1st Int. Conf. on Evolvable Systems: from Biology to Hardware (ICES96)*, Lecture Notes in Computer Science (Springer-Verlag, Heidelberg) (in press).
- Sipper, M., 1997, Evolving uniform and non-uniform cellular automata networks, in: D. Stauffer (ed.) *Annual Reviews of Computational Physics*, Vol. V (World Scientific) (in press).
- Starkweather, T., Whitley, D. and Mathias, K., 1991, Optimization using distributed genetic algorithms, in: *Parallel Problem Solving from Nature*, Lecture Notes in Computer Science, Vol. 496, H.-P. Schwefel, and R. Männer (eds.) (Springer-Verlag, Berlin) p 176.
- Tanese, R., 1987, Parallel genetic algorithms for a hypercube, in: *Proc. 2nd Int. Conf. on Genetic Algorithms*, J.J. Grefenstette (ed.) (Lawrence Erlbaum Associates) p. 177.
- Toffoli, T. and Margolus, N., 1987, *Cellular Automata Machines* (MIT Press, Cambridge, MA).
- Tomassini, M., 1993, The parallel genetic cellular automata: application to global function optimization, in: *Proc. Int. Conf. on Artificial Neural Networks and Genetic Algorithms*, R.F. Albrecht, C.R. Reeves and N.C. Steele (eds.) (Springer-Verlag, Berlin) pp. 385–391.
- Tomassini, M., 1995, A survey of genetic algorithms, in: *Annual Reviews of Computational Physics*, Vol. III, Also available as: Technical Report 95/137, Department of Computer Science, Swiss Federal Institute of Technology, Lausanne, Switzerland, D. Stauffer (ed.) (World Scientific, Singapore) pp. 87–118.
- Tomassini, M., 1996, Evolutionary algorithms, in: *Towards Evolvable Hardware*, Lecture Notes in Computer Science, Vol. 1062, E. Sanchez and M. Tomassini (eds.) (Springer-Verlag, Berlin) pp. 19–47.
- Vichniac, G.Y., Tamayo, P. and Hartman, H., 1986, Annealed and quenched inhomogeneous cellular automata. *J. Stat. Phys.* 45, 875–883.
- Wolfram, S., 1983, Statistical mechanics of cellular automata. *Rev. Mod. Phys.* 55 (3), 601–644.
- Wolfram, S., 1984, Universality and complexity in cellular automata. *Physica D* 10, 1–35.
- Wolfram, S., 1986, Random sequence generation by cellular automata. *Adv. App. Math.* 7, 123–169.