International Journal of Modern Physics C, Vol. 8, No. 5 (1997) 1013–1024

CONVERGENCE TO UNIFORMITY IN A CELLULAR AUTOMATON VIA LOCAL COEVOLUTION

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

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

Received 16 April 1997
Revised 30 May 1997

Cellular programming is a coevolutionary algorithm by which parallel cellular systems evolve to solve computational tasks. The evolving system is a massively parallel, locally interconnected grid of cells, where each cell operates according to a local interaction rule. If this rule is identical for all cells, the system is referred to as *uniform*, otherwise, it is *non-uniform*. This paper describes an experiment that addresses the following question: Employing a local coevolutionary process to solve a hard problem, known as density classification, can an optimal *uniform* solution be found? Since our approach involves the evolution of non-uniform CAs, where cellular rules are initially assigned at random, such convergence to uniformity cannot be *a priori* expected to easily emerge. The question is of both theoretical and practical interest. As for the latter, one major advantage of local evolutionary processes is their amenability to parallel implementation, using commercially available parallel machines or specialized hardware. Our experiment shows that when such local evolution is applied to the density problem, the optimal solution can be found.

Keywords: Non-uniform Cellular Automata; Cellular Programming; Evolutionary Computation.

1. Introduction

Coevolution can be defined as reciprocal evolutionary change in interacting species. The modern introduction of the term is probably due to Ref. 1, though Darwin himself wrote "of the coadaptations of organic beings to each other."[2]

An example of a coevolutionary algorithm is *cellular programming*, which involves the evolution of parallel cellular systems to solve computational tasks.[3–8] The evolving system, known as a *non-uniform cellular automaton* (CA), is a massively parallel, locally interconnected grid of cells, where each cell operates according to a local interaction rule. The uniformity property pertains to these local rules: in a

uniform CA all cells operate according to an identical rule, whereas in a non-uniform CA different cells may operate according to different rules.

Cellular programming is a coevolutionary algorithm that operates in a completely local manner, the goal being to evolve the local cellular interaction rules. Essentially, each cell contains a genome that represents its rule, specifying the cell's behavior with respect to a small local neighborhood. Initially assigned at random, these genomes locally coevolve, each cell having access only to the genomes of its immediate neighbors, with no one cell in possession of a global view of the entire grid. The process is coevolutionary due to the close interactions between the evolving cells: Changing a cell's genome — and therefore its behavior — transforms the adaptive landscapes of its neighbors, which in turn can similarly transform the landscapes of their respective neighboring cells, and so on.

The dynamical behavior of a cellular automaton may give rise to *emergent computation*, referring to the appearance of global information processing capabilities that are not explicitly represented in the system's elementary components or in their local interconnections.[3] As such, they offer an austere yet versatile model for studying natural phenomena, as well as a powerful paradigm for attaining fine-grained, massively parallel computation.

This paper describes an experiment that addresses the following question: Employing a local coevolutionary process to solve a hard problem, known as density classification, can an optimal *uniform* solution be found? Since our approach involves the evolution of non-uniform CAs, where cellular rules are initially assigned at random, such convergence to uniformity cannot be *a priori* expected to easily emerge. This question is of both theoretical and practical interest. As for the latter, one major advantage of local evolutionary processes is their amenability to parallel implementation.[3]

We begin in Sec. 2 with a brief description of cellular automata, as well as the so-called density problem, on which we shall focus our studies, evolving cellular automata to solve it. Section 3 presents the cellular programming algorithm. The experiment itself, described in Sec. 4, involves two stages: We first describe a recently discovered *uniform* cellular automaton that can perfectly solve the density problem, followed by a demonstration that coevolution via cellular programming can indeed find this provenly optimal solution. We also study the coevolutionary process. Finally, we present some concluding remarks in Sec. 5.

2. Cellular Automata and the Density Problem

The machine model we employ is based on the cellular automata (CA) model. CAs 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 current 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.[9] The cellular array (grid) is n-dimensional, where $n = 1, 2, 3$ is used in practice. In this paper we shall concentrate on one-dimensional grids, with two possible states per cell, denoted 0 and 1. In such CAs each cell is connected to r local neighbors (cells) on either side, as well as to itself, where r is a parameter referred to as the radius (thus, each cell has $2r+1$ neighbors). 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. 1). The term *configuration* refers to an assignment of states to cells in the grid.

CAs exhibit three notable features, namely, massive parallelism, locality of cellular interactions, and simplicity of basic components (cells). The machine model we employ is an extension of the original, uniform CA model, termed *non-uniform cellular automata*.[3] Such automata function in the same way as uniform ones, the only difference being in the local cellular interaction rules that need not be identical for all cells. A major problem common to such local, parallel systems is the painstaking task one is faced with in designing them to exhibit a specific behavior or solve a particular problem. This results from the local dynamics of the system, which renders the design of local interaction rules to perform global computational tasks extremely arduous.

An example of such global emergent computation, which we shall focus on in this paper, is to use a CA to determine the global density of bits in an initial state configuration. Known as the density problem, the one-dimensional, two-state CA is presented with an arbitrary initial configuration, and should converge in time to a state of all 1s if the initial configuration contains a density of 1s > 0.5, and to all 0s if this density < 0.5; for an initial density of 0.5, the CA's behavior is undefined. Spatially periodic boundary conditions are used, resulting in a circular grid (for an $r = 1$ CA this means that the leftmost and rightmost cells are connected). It has been shown that for a uniform one-dimensional grid of fixed size N, and for a fixed radius $r \geq 1$, there exists no two-state CA rule which correctly classifies all possible initial configurations;[10] this says nothing, however, about how well an imperfect CA might perform, i.e. one that does not necessarily correctly classify *all* possible configurations.

Recently, researchers have focused on the use of artificial evolution techniques, demonstrating that high-performance (though imperfect) CAs can be evolved to solve this problem. References 11 and 12 applied a standard genetic algorithm,[13] and Ref. 14 applied genetic programming[15] to evolve *uniform* CAs that exhibit high performance on the density task. We have applied *cellular programming*, described in the next section, to the coevolution of *non-uniform* CAs, the operation of one of which is demonstrated in Fig. 1.[3]

3. Cellular Programming

We study 2-state, non-uniform CAs, in which each cell may contain a different rule.

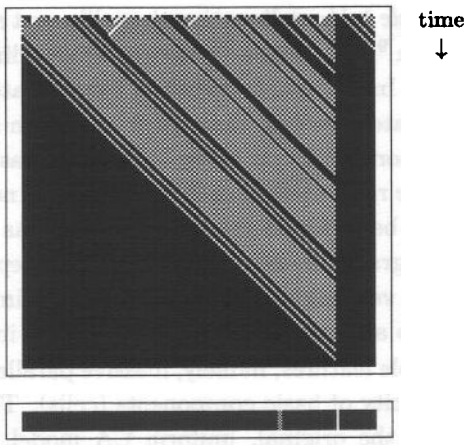

Fig. 1. The density task: Operation of a coevolved, non-uniform, $r = 1$ CA. Grid size is $N = 149$. Top figure: the pattern of configurations is shown through time (which increases down the page). White squares represent cells in state 0, black squares represent cells in state 1. Bottom figure: the rules map, depicting the distribution of rules by assigning a unique gray level to each distinct rule. Note that the evolved grid is *quasi*-uniform, meaning that the number of distinct rules is extremely small, distributed such that a subset of dominant rules occupies most of the grid (in this evolved CA, there are three distinct rules, one of which is dominant). The initial density of 1s in the random initial configuration is 0.53 and the CA relaxes to a fixed pattern of all 1s, which is the correct output.

A cell's rule table is encoded as a bit string, known as the "genome," containing the next-state (output) bits for all possible neighborhood configurations, listed in lexicographic order; e.g. for CAs with $r = 1$, the genome consists of 8 bits, where the bit at position 0 is the state to which neighborhood configuration 000 is mapped to and so on until bit 7, corresponding to neighborhood configuration 111. 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.[a] Initial configurations are then generated at random, 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 time steps, 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. 2.

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. Note that the fitness function uses the initial density to determine whether

[a]Note that our algorithm is not necessarily restricted to a single, non-uniform CA since an ensemble of distinct grids can evolve in parallel.

```
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
                             (this case can occur if the cellular radius, r, > 1)
         end if
         f_i = 0
      end parallel for
   end if
end while
```

Fig. 2. Pseudo-code of the cellular programming algorithm.

a cell is in the correct state or not at the final time step (this is also the case in the experiments described by Ref. 12). This can be compared to supervised learning in artificial neural networks, where the network is provided with correct input-output pairs during the learning phase. These samples represent, however, only a small fraction of the space of possible inputs, and the objective is for the network to generalize, i.e. correctly classify novel patterns (that were not presented during learning). This is similar to our evolutionary process, where the cellular automaton is presented with only a minute fraction of the possible input configurations during evolution, and is expected thereafter to correctly classify previously unseen patterns.

There are two main differences between our algorithm and the standard genetic algorithm:[13]

(1) The latter involves a population of evolving, uniform CAs, with all individuals *ranked* according to fitness, and 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. This characteristic is of prime import where, for example, hardware implementation is concerned.[3]

(2) The standard genetic algorithm involves a population of *independent* problem solutions, meaning that 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 *coevolves* since each cell's fitness depends upon its evolving neighbors. This may also be considered a form of symbiotic cooperation, which falls, as does coevolution, under the general heading of "ecological" interactions (see Ref. 13, pages 182–183).

Much of the work on cellular programming is described in the recent book by Sipper[3] (see also reviews in Refs. 5 and 6, as well as the web page http://lslwww.epfl.ch/~moshes/cp.html).

4. Convergence to Uniformity via Cellular Programming

The cellular programming algorithm involves the coevolution of a non-uniform CA. Thus, it embodies a process by which an initially random cellular system, where each cell behaves according to a randomly chosen rule, coevolves to solve a given task. In our previous work we noted that *quasi*-uniform CAs had thus emerged, where the number of distinct rules is extremely small, distributed such that a subset of dominant rules occupies most of the grid.[3] This is demonstrated in Fig. 1 via the rules map, depicting the distribution of rules by assigning a unique gray level to each distinct rule.

The central question addressed in the experiment described below is the following: Can a local coevolutionary process, that starts out with a random cellular system (which is therefore completely non-uniform), find a globally optimal solution, which involves a completely uniform system (i.e. with an identical rule present in each cell)? To study this problem we proceed in two stages: First, we describe a recently discovered *uniform* CA that can perfectly solve the density problem; then, we show that coevolution via cellular programming can indeed find this provenly optimal solution.

4.1. *A CA that solves the density problem*

As explained in Sec. 2, the density problem specifies that the one-dimensional, two-state CA, upon presentation of an arbitrary initial configuration, should converge in time to a state of all 1s if the initial configuration contains a density of 1s > 0.5, and to all 0s if this density < 0.5; for an initial density of 0.5, the CA's behavior is undefined. Spatially periodic boundary conditions are used, resulting in a circular grid. As noted, it has been shown that for a uniform one-dimensional grid of fixed

size N, and for a fixed radius $r \geq 1$, there exists no two-state CA rule which correctly classifies all possible initial configurations.[10]

The density problem studied to date specifies convergence to one of two fixed-point configurations, which are considered as the output of the computation. Recently, Ref. 16 showed that a perfect CA density classifier exists, upon defining a different output specification. Consider the uniform, two-state, $r = 1$ rule-184 CA,[b] defined as follows:

$$s_i(t+1) = \begin{cases} s_{i-1}(t) & \text{if } s_i(t) = 0, \\ s_{i+1}(t) & \text{if } s_i(t) = 1 \end{cases}$$

where $s_i(t)$ is the state of cell i at time t. Upon presentation of an arbitrary initial configuration, the grid relaxes to a limit-cycle, within $\lceil N/2 \rceil$ time steps, that provides a classification of the initial configuration's density of 1s: if this density > 0.5 (respectively, < 0.5), then the final configuration consists of one or more blocks of at least two consecutive 1s (0s), interspersed by an alternation of 0s and 1s; for an initial density of exactly 0.5, the final configuration consists of an alternation of 0s and 1s. The computation's output is given by the state of the consecutive block (or blocks) of same-state cells (Fig. 3). Reference 16 proved the following theorem concerning rule 184: For a finite-size CA of size N, let $S(t) = \{s_0(t), \ldots, s_{N-1}(t)\}$ be the grid configuration at time step t, let $D(\{s_i(t), \ldots, s_{i+k-1}(t)\})$ be the density of 1s at time t over a block of k cells at positions $\{i, \ldots, i+k-1\}$, and let $T = \lceil N/2 \rceil$ (cellular indices are computed modulus the grid size N since the grid is circular). Then:

1. If $D(S(0)) > 0.5$ then: (a) there exists a pair of adjacent cells $i, i+1$ such that $s_i(T) = 1$ and $s_{i+1}(T) = 1$, and (b) for all i, $s_i(T) = 0 \Rightarrow s_{i+1}(T) = 1$.
2. If $D(S(0)) < 0.5$ then: (a) there exists a pair of adjacent cells $i, i+1$ such that $s_i(T) = 0$ and $s_{i+1}(T) = 0$, and (b) for all i, $s_i(T) = 1 \Rightarrow s_{i+1}(T) = 0$.
3. If $D(S(0)) = 0.5$ then for all i, $s_i(T) \neq s_{i+1}(T)$.

Thus, rule 184 performs perfect density classification (including the density $= 0.5$ case). We note in passing that the reflection-symmetric rule 226 holds the same properties of rule 184.

As the input configuration is random, this entails a high Kolmogorov complexity. Intuitively, for a given finite string, this measure concerns the size of the shortest program that computes the string.[18] Both the fixed-point output of the original problem, as well as the novel "blocks" output, involve a notable reduction with respect to this complexity measure. It has been noted in Ref. 12 that the computational complexity of the input is that of a non-regular language[19] since a counter register is needed, whose size is proportional to $\log(N)$, whereas the fixed-point output of the original problem involves a simple regular language (all 0s or all 1s); we note that the novel output specification also involves a regular language (a block of

[b]Rule numbers are given in accordance with Wolfram's convention,[17] representing the decimal equivalent of the binary number encoding the rule table. For example, rule 184, or 10111000 in its binary form, maps the local neighborhoods $111 \mapsto 1$, $110 \mapsto 0, \ldots$, $000 \mapsto 0$.

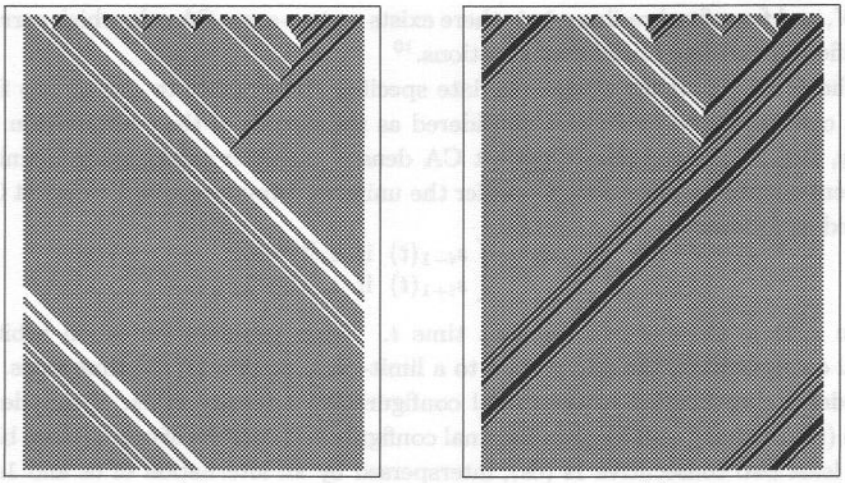

Fig. 3. Density classification: Demonstration of the uniform rule-184 CA on two random initial configurations. Grid size is $N = 149$. The pattern of configurations is shown for the first 200 time steps. (a) Initial density is 0.47. The final configuration consists of an alternation of 0s and 1s with several blocks of two or more cells in state 0. (b) Initial density is 0.53. The final configuration consists of an alternation of 0s and 1s with several blocks of two or more cells in state 1. In both cases the CA correctly classifies the initial configuration.

two state-0 or state-1 cells). Reference 16 thus concluded that their newly proposed density classifier is as viable as the original one with respect to these complexity measures, while surpassing the latter in terms of performance.

4.2. *The experiment and its results*

Having proven that there exists a CA that perfectly solves the density problem, we now ask whether this CA can evolve via cellular programming. This means that, setting out with a completely non-uniform CA, the evolutionary process must converge to a completely uniform one, where all cells contain the same rule. As with any other artificial evolution technique, a critical factor is the fitness function. The one used by us for the rule-184 experiment was obtained by observing the "signals" in Fig. 3, i.e. the propagation of cellular states through time (for a more formal definition of signals in CAs see Ref. 20). Note that in a typical application of an evolutionary algorithm (or any search methodology for that matter) a viable solution is not available in advance for examination. However, our experiment is not meant to discover a novel solution, but rather to seek whether local evolution can converge towards a known optimal one. Let $D(t)$ denote the density of 1s over the entire grid at time step t. Referring to Fig. 2, cell i is considered to be in the correct final state, after being run for M time steps on a random initial configuration (thereby having its fitness value, f_i, incremented), if one of the following four cases holds true:

1. $D(0) < 0.5$ and $s_i(M) = 0$ and $s_{i-1}(M-1) = 0$,
2. $D(0) < 0.5$ and $s_{(i-1,i,i+1)}(M) = (0,1,0)$ and $s_{(i-2,i-1,i)}(M-1) = (0,1,0)$,
3. $D(0) > 0.5$ and $s_i(M) = 1$ and $s_{i+1}(M-1) = 1$,
4. $D(0) > 0.5$ and $s_{(i-1,i,i+1)}(M) = (1,0,1)$ and $s_{(i,i+1,i+2)}(M-1) = (1,0,1)$,

where $s_{(i,j,k)}(t)$ are the states of cells i, j, k at time t. For example, the first case implies that when there is a majority of 0s in the initial configuration, a state of 0 "propagates" to the right (e.g. Fig. 3(a)). If none of the above four cases holds true after M time steps, then the cell's fitness score for this initial configuration is 0, i.e. its fitness value f_i is not incremented.

Our experiment consisted of 100 evolutionary runs of the cellular programming algorithm, each run for a given maximal number of initial configurations (the evolutionary run was terminated when this maximum was reached, or upon convergence to rule 184). Six such experiments were conducted, results of which are presented in Table 1. The most notable feature is that convergence to a uniform grid, in which all cells contain rule 184, occurs in a majority of the runs. Not surprisingly, convergence rate increases as the maximal number of configurations is increased. Both odd- and even-sized grids were employed, with no observable differences between them. Note that given the grid sizes used and the small number of possible rules (256), it is quite possible that rule 184 be found in the initial grid (or at least a rule that is off by only a 1-bit mutation). However, it is not obvious *a priori* that the selection forces at work can cause this rule to diffuse throughout the grid. As noted, the process is coevolutionary, with the cells interacting and thus influencing each other's fitness. While rule 184 may find its way into the grid quite quickly, it is far from evident what the optimal *population* of rules is, and how to find it. Nonetheless, through the local evolutionary process described herein, the optimal, uniform grid emerges.

While a majority of the evolutionary runs "found" the optimal uniform CA, others resulted in non-uniform ones which are not optimal. An interesting regularity in

Table 1. Results of experiments. Shown below are the CA parameters (grid size N and number of time steps M), the number of evolutionary runs, the maximal number of initial configurations per run, and the number of times the final evolved grid was a uniform one, in which all cells contain rule 184.

N (Grid size)	M (Time steps)	No. evol. runs	Max. configs. per run	No. converg. to rule 184
101	51	100	100,000	44
101	51	100	250,000	65
120	60	100	200,000	45
120	60	100	400,000	58
149	75	100	250,000	66
149	75	100	500,000	73

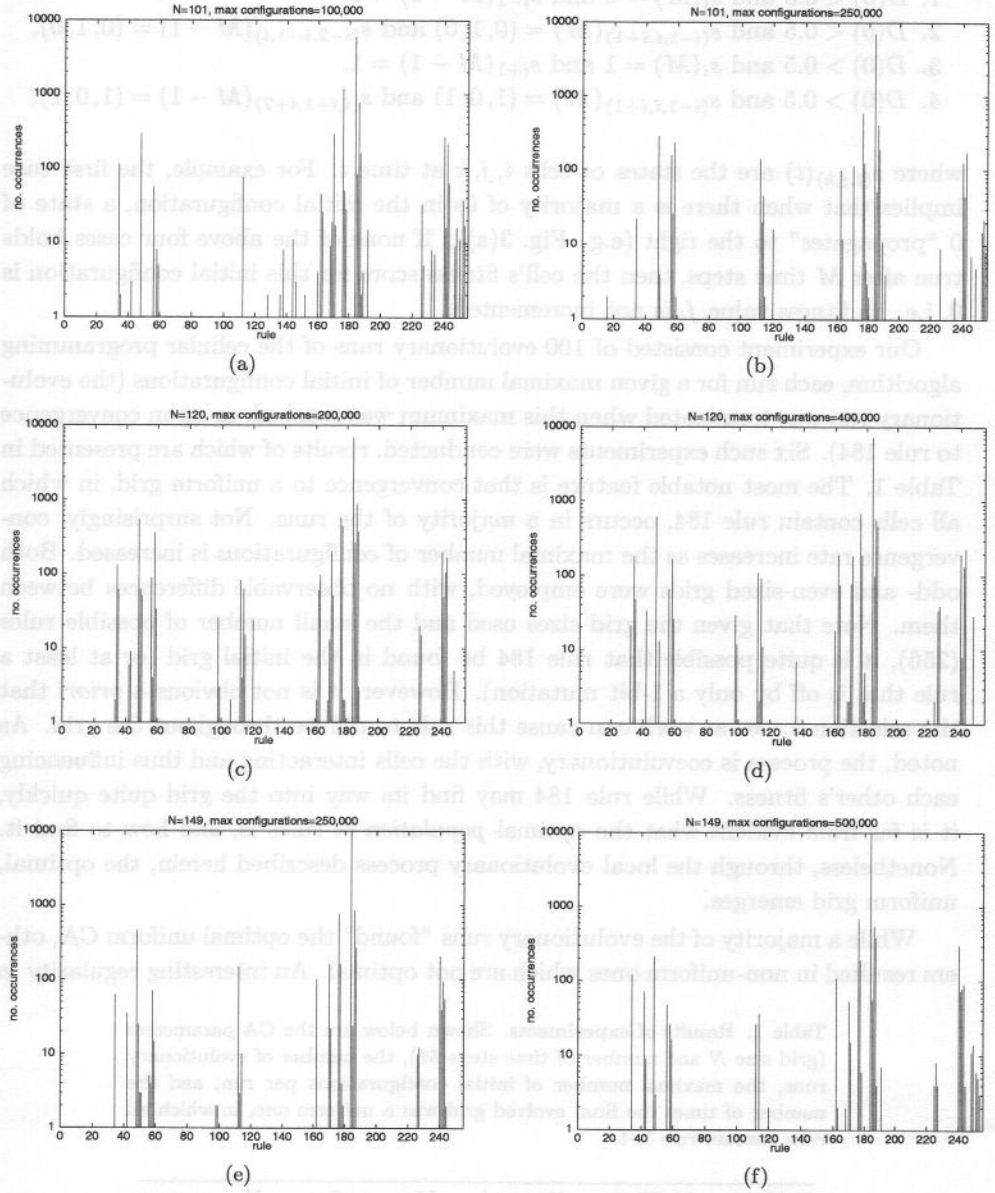

Fig. 4. Rules present in the final evolved grids. For each of the six experiments, consisting of 100 evolutionary runs, the respective histogram shows the number of occurrences of each of the possible 256 rules.

the evolutionary process was revealed when examining the ensemble of rules present in the evolved grids. Our results are presented in Fig. 4 in the form of histograms, showing the number of occurrences of each of the possible 256 rules within the final evolved grids. The manifest similarity between the six experiments suggests that the coevolutionary fitness landscape is independent of the CA parameters (N, M), the presence of some rules proving advantageous, the presence of others proving detrimental.

The $r = 1$ rules discussed herein can be expressed in minimal disjunctive normal form, using boolean multiplication and addition (corresponding to the AND and OR operations), and bar to denote complementation.[17] Observing the histograms in Fig. 4, we find that the most widespread rules are (in decreasing order of appearance):

rule	minimal disjunctive normal form
184	$s_i(t+1) = s_{i-1}(t)\overline{s_i}(t) + s_i(t)s_{i+1}(t)$
176	$s_i(t+1) = s_{i-1}(t)\overline{s_i}(t) + s_{i-1}(t)s_{i+1}(t)$
186	$s_i(t+1) = s_{i-1}(t)\overline{s_i}(t) + s_{i+1}(t)$
48	$s_i(t+1) = s_{i-1}(t)\overline{s_i}(t)$
187	$s_i(t+1) = \overline{s_i}(t) + s_{i+1}(t)$

We note that the top-ranking evolved rules are similar to rule 184. It was also observed that rule 184's reflection-symmetric rule, 226, appeared quite sparsely. As noted above, this rule holds the same properties as rule 184, however, the "signals" are "reversed." Its sparse occurrence is evidence of the fitness function's precision, "targeting" exactly those signals manifest by rule 184 (Fig. 3).

5. Concluding Remarks

We set out to investigate whether the local coevolutionary cellular programming algorithm can find a provenly optimal uniform CA that solves the density problem. Since our approach involves the evolution of non-uniform CAs, where cellular rules are initially assigned at random, such convergence to uniformity cannot be *a priori* expected to easily emerge. Our experiment shows that in the case of the density problem the optimal solution can be found. Although this result does not necessarily apply to local coevolution in general, it is nonetheless an encouraging step. It should be mentioned that no evolutionary algorithm is guaranteed to produce an optimal solution (or even a good one) for a general search problem.[13] As noted, one motivation for such studies is the observation that local evolutionary processes are more amenable to parallel implementation, using commercially available parallel machines or specialized hardware.

References

1. C. J. Mode, *Evolution* **12**, 158 (1958).

2. C. Darwin, *The Origin of Species* (John Murray, London, 1859).
3. M. Sipper, *Evolution of Parallel Cellular Machines: The Cellular Programming Approach* (Springer-Verlag, Heidelberg, 1997).
4. M. Sipper, *Physica* **D92**, 193 (1996).
5. M. Sipper, "Evolving uniform and non-uniform cellular automata networks," in *Annual Reviews of Computational Physics*, Volume V, ed. D. Stauffer (World Scientific, Singapore, 1997), pp. 243–285.
6. M. Sipper, *BioSystems*, **42**, 29 (1997).
7. M. Sipper and E. Ruppin, *Physica* **D99**, 428 (1997).
8. M. Sipper and M. Tomassini, *Int. J. Mod. Phys.* **C7**(2), 181 (1996).
9. T. Toffoli and N. Margolus, *Cellular Automata Machines* (MIT Press, Cambridge, 1987).
10. M. Land and R. K. Belew, *Phys. Rev. Letts.* **74**(25), 5148 (1995).
11. N. H. Packard, "Adaptation toward the edge of chaos," in *Dynamic Patterns in Complex Systems*, eds. J. A. S. Kelso, A. J. Mandell, and M. F. Shlesinger (World Scientific, 1988), pp. 293–301.
12. M. Mitchell, J. P. Crutchfield, and P. T. Hraber, *Physica* **D75**, 361 (1994).
13. M. Mitchell, *An Introduction to Genetic Algorithms* (MIT Press, Cambridge, MA, 1996).
14. D. Andre, F. H Bennett III, and J. R. Koza, "Discovery by genetic programming of a cellular automata rule that is better than any known rule for the majority classification problem," in eds. J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, *Genetic Programming 1996: Proc. First Ann. Conf.* (MIT Press, Cambridge, MA, 1996), pp. 3–11.
15. J. R. Koza, *Genetic Programming* (MIT Press, Cambridge MA, 1992).
16. M. S. Capcarrere, M. Sipper, and M. Tomassini, *Phys. Rev. Lett.* **77**(24), 4969 (1996).
17. S. Wolfram, *Cellular Automata and Complexity* (Addison-Wesley, Reading, MA, 1994).
18. M. Li and P. Vitányi, *An Introduction to Kolmogorov Complexity and its Applications* (Springer-Verlag, New York, 1993).
19. J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory Languages and Computation* (Addison-Wesley, Redwood City, CA, 1979).
20. J. P. Crutchfield and M. Mitchell, "The evolution of emergent computation," *Proc. of the Nat. Acad. of Sci. USA*, **92**(23), 10742 (1995).