



ELSEVIER

Physica D 99 (1997) 428–441

PHYSICA D

Co-evolving architectures for cellular machines

Moshe Sipper^{a,*}, Eytan Ruppin^{b,1}

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

^b *Department of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel*

Received 2 November 1995; revised 13 March 1996; accepted 4 June 1996

Communicated by H. Flaschka

Abstract

Recent studies have shown that *non-uniform* cellular automata (CA), where cellular rules need not necessarily be identical, can be co-evolved to perform computational tasks. This paper extends these studies by generalizing on a second aspect of CAs, namely their standard, homogeneous connectivity. We study *non-standard* architectures, where each cell has a small, identical number of connections, yet not necessarily from its most immediate neighboring cells. We show that such architectures are computationally more efficient than standard architectures in solving global tasks, and also provide the reasoning for this. It is shown that one can successfully *evolve* non-standard architectures through a two-level evolutionary process, in which the cellular rules evolve concomitantly with the cellular connections.

Specifically, studying the global *density* task, we identify the average cellular distance as a prime architectural parameter determining cellular automata performance. We carry out a quantitative analysis of this relationship, our main results being: (1) performance is *linearly* dependent on the average cellular distance, with a high correlation coefficient; (2) high performance architectures can be co-evolved, concomitantly with the rules, and (3) low connectivity cost can be obtained as well as high performance.

The evolutionary algorithm presented may have important applications to designing economical connectivity architectures for distributed computing systems.

1. Introduction

Cellular automata (CA) are dynamical systems in which space and time are discrete. They consist 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 [21,23].

CAs exhibit three notable features, namely massive parallelism, locality of cellular interactions, and simplicity of basic components (cells). They perform computations in a distributed fashion on a spatially extended grid; as such they differ from the standard approach to parallel computation in which a problem is split into independent sub-problems, each solved by a different processor, later to be combined in order to yield the final solution. CAs suggest a new approach in which complex behavior arises in a bottom-up manner from non-linear, spatially extended, local interactions [14].

* Corresponding author. E-mail: moshe.sipper@di.epfl.ch.

¹ E-mail: ruppin@math.tau.ac.il.

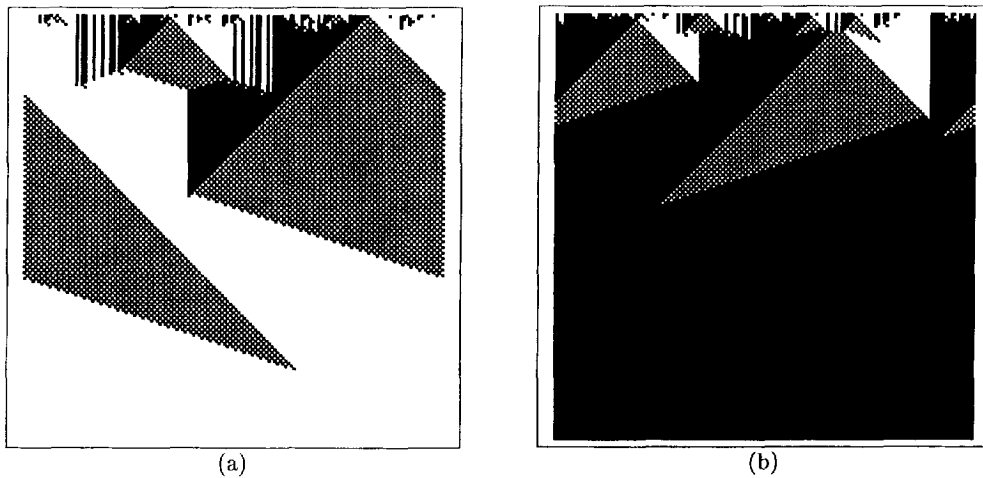


Fig. 1. The density task: operation of the GKL rule. CA is one-dimensional, uniform, two-state, with connectivity radius $r = 3$. Grid size is $N = 149$. White squares represent cells in state 0, black squares represent cells in state 1. The pattern of configurations is shown through time (which increases down the page). (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.

A major impediment preventing ubiquitous computing with CAs stems from the difficulty of utilizing their complex behavior to perform useful computations. The difficulty of designing CAs to have a specific behavior or perform a particular task has limited their applications; automating the design process would greatly enhance the viability of CAs [14].

Recent studies have shown that CAs can be evolved to perform non-trivial computational tasks. One such task, which we study in detail in this paper, is that of density classification. In this task the two-state CA must decide whether or not the initial configuration contains more than 50% 1s, where the term “configuration” refers to an assignment of 1 states to several cells, and 0s otherwise. The desired behavior (i.e., the result of the computation) is for the CA to relax to a fixed-point pattern of all 1s if the initial density of 1s exceeds 0.5, and all 0s otherwise (Fig. 1).

The density task was studied by Mitchell et al. [14,15] and Das et al. [6], who demonstrated that high performance CA rules can be evolved using genetic algorithms. We have investigated an extension of the CA model termed *non-uniform cellular automata*, in which cellular rules need not be identical [17–19]. Employing this model we found that high performance can be attained for the density task by

means of co-evolution [20]². Non-uniform CAs have also been investigated by Vichniac et al. [22] and Hartman and Vichniac [10].

As noted by Mitchell et al., density is a global property and hence the task comprises a non-trivial computation for a locally connected CA. Since the 1s can be distributed throughout the grid, propagation of information must occur over large distances (i.e., $O(N)$). The computation involved corresponds to recognition of a non-regular language, since the minimum amount of memory required for the task is $O(\log N)$ using a serial scan algorithm [4–6,13–16]. Note that the density task cannot be perfectly solved by a uniform, two-state CA, as recently proven by Land and Belew [12]; however, no upper bound is currently available on the best possible imperfect performance, attained to date by the Gacs–Kurdyumov–Levin (GKL) rule [7,9] (Fig. 1).

Previous studies of the density task were conducted using locally connected, one-dimensional grids [14,20]. The task can be extended in a straightforward manner to two-dimensional grids, an investigation of which we have carried out, using the same number of local connections per cell as in the one-dimensional

² A precise definition of the performance measure is given in Section 4.

case. We found that markedly higher performance is attained for the density task with two-dimensional grids along with shorter computation times. This finding is intuitively understood by observing that a two-dimensional, locally connected grid can be embedded in a one-dimensional grid with local and distant connections. This can be achieved, for example, by aligning the rows of the two-dimensional grid so as to form a one-dimensional array; the resulting embedded one-dimensional grid has distant connections of order \sqrt{N} , where N is the grid size. 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.

Motivated by this observation concerning the effect of connection lengths on performance, our primary goal in this paper is to quantitatively study the relationship between performance and connectivity on a global task, in one-dimensional CAs. The main contribution of this paper is identifying the average cellular distance (see Section 2) as the prime architectural parameter which linearly determines CA performance. We find that high performance architectures can be co-evolved concomitantly with the rules, and that it is possible to evolve such architectures that exhibit low connectivity cost as well as high performance. This work extends our previous work on the co-evolution of non-uniform CAs [20] by studying *evolving architectures*. Our motivation stems from two primary sources: (a) finding more efficient CA architectures via evolution, (b) the co-evolution of architectures offers a promising approach for solving a general wiring problem for a set of distributed processors, subject to given constraints. The efficient solution of the density task by CAs with evolving architectures may have important applications to designing efficient distributed computing networks.

In Section 2 we describe the CA architectures studied in this work. In Section 3 we describe the cellular programming algorithm used to co-evolve non-uniform CAs. Section 4 discusses CA rule evolution with fixed architectures. In Section 5 we extend our evolutionary algorithm such that the architecture evolves as well as the cellular rules. In Section 6 we

study the evolution of low cost architectures. Our findings and their possible future application to designing distributed computer networks are discussed in Section 7.

2. Architecture considerations

We use the term *architecture* to denote the connectivity pattern of CA cells. In the standard one-dimensional model a cell is connected to r local neighbors on either side as well as to itself, where r is referred to as the radius (thus each cell has $2r + 1$ neighbors). The model we consider is that of non-uniform CAs with non-standard architectures, in which cells need not necessarily contain the same rule nor be locally connected; however, as with the standard CA model, each cell has a small, identical number of impinging connections. In what follows the term *neighbor* refers to a directly connected cell. We shall employ the cellular programming algorithm to evolve cellular rules for non-uniform CAs whose architectures are fixed (yet non-standard) during the evolutionary run, or evolve concomitantly with the rules; these are referred to as fixed or evolving architectures, respectively.

We consider one-dimensional, symmetrical architectures where each cell has four neighbors, with connection lengths of a and b , as well as a self-connection. Spatially periodic boundary conditions are used, resulting in a circular grid (Fig. 2). This type of architecture belongs to the general class of

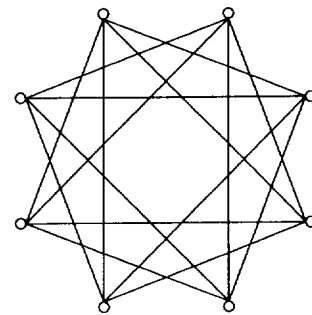


Fig. 2. A $C_8(2, 3)$ circulant graph. Each node is connected to four neighbors, with connection lengths of 2 and 3.

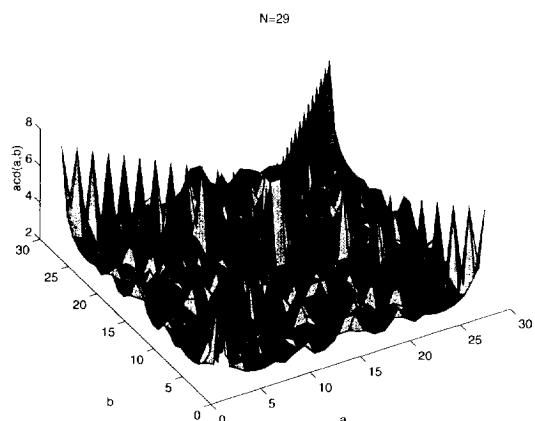


Fig. 3. The ruggedness of the acd landscape is illustrated by plotting it as a function of connection lengths (a, b) for grids of size $N = 29$. Each (a, b) pair entails a different $C_{29}(a, b)$ architecture whose acd is represented as a point in the graph.

circulant graphs [3]: For a given positive integer N , let n_1, n_2, \dots, n_k be a sequence of integers where

$$0 < n_1 < n_2 < \dots < n_k < \frac{1}{2}(N + 1).$$

Then the *circulant graph* $C_N(n_1, n_2, \dots, n_k)$ is the graph on N nodes v_1, v_2, \dots, v_N with node v_i connected to each node $v_{i \pm n_j \pmod{N}}$. The values n_j are referred to as *connection lengths*. The *distance* between two cells on the circulant is the number of connections one must traverse on the shortest path connecting them. The architectures studied here are circulants $C_N(a, b)$.

We surmise that attaining high performance on global tasks requires rapid information propagation throughout the CA, and that the rate of information propagation across the grid inversely depends on the average cellular distance (acd). Before proceeding to study performance, let us examine how the acd of a $C_N(a, b)$ architecture varies as a function of (a, b) . As shown in Fig. 3, the acd landscape is extremely rugged (the algorithm used to calculate the acd is described in Appendix A). This is due to the relationship between a and b – if $\text{gcd}(a, b) \neq 1$ the acd is markedly higher than when $\text{gcd}(a, b) = 1$ (note that the circulant graph $C_N(n_1, n_2, \dots, n_k)$ is connected if and only if $\text{gcd}(n_1, n_2, \dots, n_k, N) = 1$ [1]).

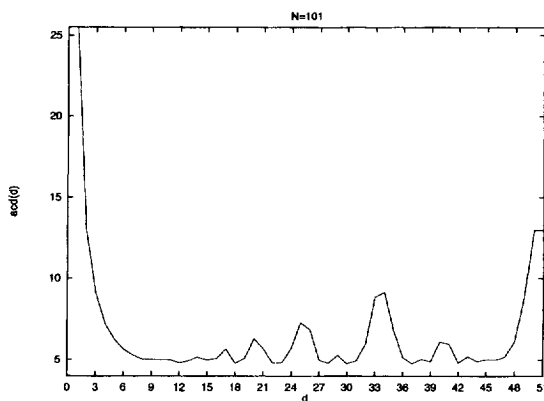


Fig. 4. $C_{101}(1, d)$: average cellular distance (acd) as a function of d . acd is plotted for $d \leq \frac{1}{2}N$, as it is symmetric about $d = \frac{1}{2}N$.

It is straightforward to show that every $C_N(a, b)$ architecture is isomorphic to a $C_N(1, d')$ architecture, for some d' , referred to as the *equivalent d'* (see Appendix A). Graph $C_N(a, b)$ is isomorphic to a graph $C_N(1, d')$ if and only if every pair of nodes linked via a connection of length a in $C_N(a, b)$ is linked via a connection of length 1 in $C_N(1, d')$, and every pair linked via a connection of length b in $C_N(a, b)$ is linked via a connection of length d' in $C_N(1, d')$ ³. We may therefore study the performance of $C_N(1, d)$ architectures, our conclusions being applicable to the general $C_N(a, b)$ case; this is important from a practical standpoint since the $C_N(a, b)$ architecture space is extremely large. However, if one wishes to minimize connectivity *cost*, defined as $a + b$, as well as to maximize performance, general $C_N(a, b)$ architectures must be considered; the equivalent d' value of a $C_N(a, b)$ architecture may be large, resulting in a lower cost of $C_N(a, b)$ as compared with the isomorphic $C_N(1, d')$ architecture (for example, the equivalent of $C_{101}(3, 5)$ is $C_{101}(1, 32)$).

Fig. 4 depicts the acd for $C_N(1, d)$ architectures, $N = 101$. It is evident that the acd varies considerably as a function of d ; as d increases from $d = 1$ the acd

³ This is not necessarily a one-to-one mapping; $C_N(a, b)$ may map to $C_N(1, d'_1)$ and $C_N(1, d'_2)$, however, we select the minimum of d'_1 and d'_2 , thus obtaining a unique mapping.

declines and reaches a minimum at $d = O(\sqrt{N})$. This supports the notion put forward in Section 1 concerning the advantage of two-dimensional grids.

We concentrate on the following issues:

- (1) How strongly does the d determine performance on global tasks?
- (2) Can high performance architectures be evolved, that is can “good” d or (a, b) values be discovered through evolution?
- (3) Can high performance architectures be co-evolved, that exhibit low connectivity cost as well?

3. The cellular programming algorithm

We study two-state, one-dimensional, 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 next-state bits for all possible neighborhood configurations; e.g., for CAs with $r = 2$, the genome consists of 32 bits, where the bit at position 0 is the state to which neighborhood configuration 00000 is mapped to and so on until bit 31 corresponding to neighborhood configuration 11111. 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 , where cell rules are initialized at random. Initial configurations are generated at random, uniformly distributed over densities in the range $[0.0, 1.0]$. For each initial configuration the CA is run for M time steps (in our simulations we used $M = N$ so that computation time is linear with grid size). 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 Fig. 5. In our simulations, the total number of

initial configurations per evolutionary run was in the range $[50\,000, 500\,000]$ ⁴.

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 evolutionary algorithm and that used by Mitchell et al.: (a) In their work, a standard genetic algorithm is used, employing a population of evolving, uniform CAs. All CAs are *ranked* according to fitness, with crossover occurring between *any* two CA rules. 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; each CA is run independently, after which genetic operators are applied to produce a new population. In contrast, our CA *co-evolves* since each cell’s fitness depends upon its evolving neighbors.

4. Fixed architectures

In this section we study the effects of different architectures on performance, by applying the cellular programming algorithm to the evolution of cellular rules using fixed, non-standard architectures. We performed numerous evolutionary runs using $C_N(1, d)$ architectures with different values of d , recording the maximal performance attained during the run; *performance* is defined as the average fitness of all grid cells over the last C configurations, normalized to the range $[0.0, 1.0]$. Before proceeding, we point out that this is

⁴ By comparison, Mitchell et al. employed a genetic algorithm with a population size of 100, which was run for 100 generations; every generation each CA was run on 100–300 initial configurations, resulting in a total of $[10^6, 3 \times 10^6]$ configurations per evolutionary run.

```

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
  end if
   $f_i = 0$ 
end parallel for
end if
end while

```

Fig. 5. Pseudo-code of the cellular programming algorithm.

somewhat different than the work of Mitchell et al., who defined three measures: (1) performance – the number of correct classifications on a sample of initial configurations, randomly chosen from a binomial distribution over initial densities, (2) performance fitness – the number of correct classifications on a sample of C initial configurations chosen from a uniform distribution over densities in the range $[0.0, 1.0]$ (no partial credit is given for partially correct final configurations), and (3) proportional fitness – the fraction of cell states correct at the last iteration, averaged over C initial configurations, uniformly distributed over densities in the range $[0.0, 1.0]$ (partial credit is given). Our performance measure is analogous to the latter measure, however, there is an important difference: as our evolutionary algorithm is local, fitness values are computed for each individual cell; global fitness of the CA can then be *observed* by averaging these values over the entire grid. As for the choice of initial configurations, Mitchell et al. remarked that the binomial

distribution is more difficult than the uniform-over-densities one since the former results in configurations with a density in the proximity of 0.5, thereby entailing harder correct classification. This distinction did not prove essential in our studies since our focus is on the relationship between performance and connectivity on a global task, toward which end we selected the uniform-over-densities distribution as a benchmark measure by which to evolve CAs and compare their performance. We shall, nonetheless, demonstrate that our CAs attain high performance even when applying the binomial distribution.

Fig. 6 depicts the results of our evolutionary runs, along with the acd graph. Markedly higher performance is attained for values of d corresponding to low acd values and vice versa. While performance behaves in a rugged, non-monotonic manner as a function of d , it is linearly correlated with acd (with a correlation coefficient of 0.99, and a negligible p value) as depicted in Fig. 7.

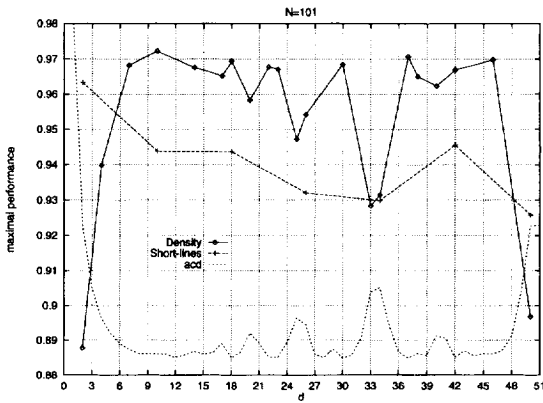


Fig. 6. $C_{101}(1, d)$: maximal evolved performance on the density and short-lines tasks as a function of d . The graph represents the average results of 420 evolutionary runs; 21 d values were tested for the density task and seven for the short-lines task. For each such d value, 15 evolutionary runs were performed with 50 000 initial configurations per run. Each graph point represents the average value of the respective 15 runs; standard deviations of these averages are in the range 0.003–0.011. i.e., 3–11% of the performance range in question (deviations were computed excluding the two extremal values).

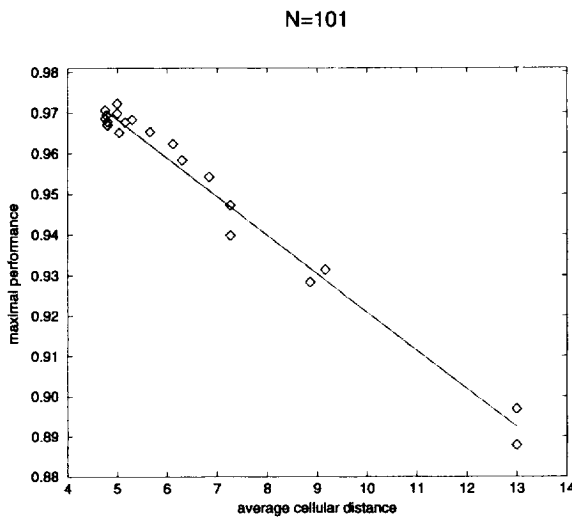


Fig. 7. $C_{101}(1, d)$: maximal performance on the density task as a function of average cellular distance. The linear regression shown has a correlation coefficient of 0.99, with a p value that is practically zero.

How does the architecture influence performance when the CA is evolved to solve a local task? To test this we introduced the short-lines task: given an initial configuration consisting of five non-filled intervals of

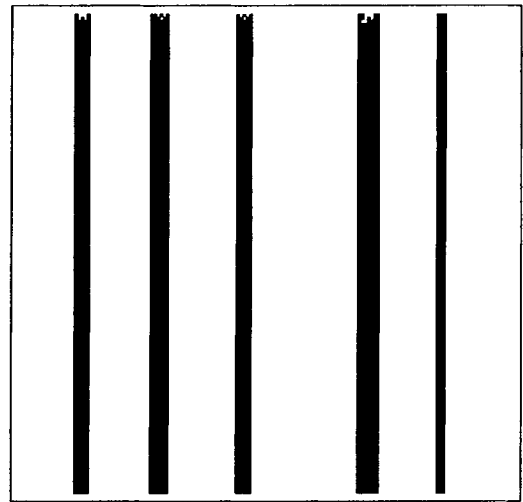


Fig. 8. The short-lines task: operation of a co-evolved, non-uniform CA of size $N = 149$ with a standard architecture of connectivity radius $r = 2$ ($C_{149}(1, 2)$).

random length between 1–7, the CA must reach a final configuration in which the intervals form continuous lines (Fig. 8). In this final configuration all cells within the confines of an interval should be in state 1, and all other cells should be in state 0 (in our simulations, cells within an interval in the initial configuration were set to state 1 with probability 0.3; cells outside an interval were set to 0). Fig. 6 demonstrates that performance for this local task is maximal for minimal d , and decreases as d increases.

These results demonstrate that performance is strongly dependent upon the architecture, with higher performance attainable by using different architectures than that of the standard CA model. We also observe that the global and local tasks studied have different efficient architectures.

As each $C_N(a, b)$ architecture is isomorphic to a $C_N(1, d)$ one, and since performance is correlated with acd in the $C_N(1, d)$ case, it follows that the performance of general $C_N(a, b)$ architectures is also correlated with acd. It is interesting to note the ruggedness of the equivalent d' landscape, depicted in Fig. 9, representing the equivalent d' value for each (a, b) pair. Table 1 presents the performance results of four $C_N(a, b)$ architectures on the density task: $C_{101}(3, 5)$, $C_{102}(3, 5)$, $C_{101}(3, 6)$ and $C_{102}(3, 6)$, demonstrating

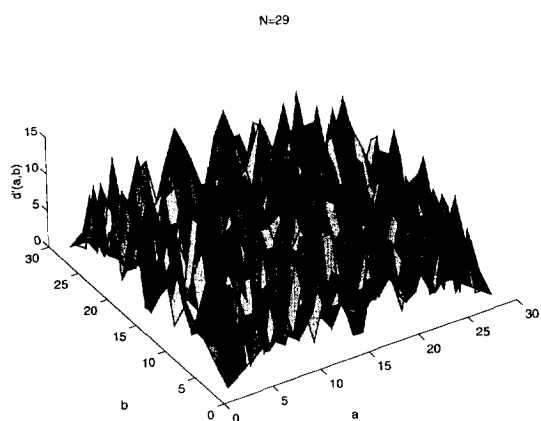


Fig. 9. The ruggedness of the equivalent d' landscape is illustrated by plotting it as a function of (a, b) , for $C_{29}(a, b)$.

Table 1
Maximal evolved performance for $C_N(a, b)$ on the density task

(a, b)	N	acd	Equivalent d'	Mean maximal performance
(3, 5)	101	5.98	32	0.96 (0.006)
(3, 5)	102	6.02	21	0.96 (0.005)
(3, 6)	101	13	2	0.88 (0.01)
(3, 6)	102	Not connected	None	0.75 (0.07)

For each architecture, 15 evolutionary runs were performed with 50 000 initial configurations per run. The average maximal performance attained on these runs is shown along with standard deviations in parentheses (deviations were computed excluding the two extremal values).

the dependence on the acd. Since $\text{gcd}(3, 5) = 1$ whereas $\text{gcd}(3, 6) \neq 1$ (resulting in a lower acd for architectures with the former connectivity), we find, as expected, that $C_N(3, 5)$ exhibits significantly higher performance than $C_N(3, 6)$. Furthermore, since $C_{102}(3, 6)$ is not a connected graph (see Section 2), this architecture displays even lower performance. The operation of a co-evolved, $C_{149}(3, 5)$ CA on the density task is demonstrated in Fig. 10.

5. Evolving architectures

In Section 4 we employed the cellular programming algorithm to evolve non-uniform CAs with fixed $C_N(a, b)$ or $C_N(1, d)$ architectures. We concluded that judicious selection of (a, b) or d can notably increase performance, which is highly correlated with

the average cellular distance. The question we now pose is whether a priori specification of the connectivity parameters is indeed necessary or can an efficient architecture co-evolve along with the cellular rules. Moreover, can heterogeneous architectures, where each cell may have different d_i or (a_i, b_i) connection lengths, achieve high performance? Below we denote by $C_N(1, d_i)$ and $C_N(a_i, b_i)$ heterogeneous architectures with one or two evolving connection lengths per cell, respectively. Note that these are the cell's input connections, on which information is received; as connectivity is heterogeneous, input and output connections may be different, the latter specified implicitly by the input connections of the neighboring cells.

In order to evolve the architecture as well as the rules the algorithm presented in Section 3 is modified; each cell maintains a “genome” consisting of two “chromosomes”. The first, encoding the rule table, is identical to that delineated in Section 3. The second chromosome encodes the cell's connections as Gray code bit strings [11]⁵. In what follows we use grids of size $N = 129$; thus, the architecture chromosome contains six bits for evolving $C_{129}(1, d_i)$ architectures and 12 bits for $C_{129}(a_i, b_i)$ architectures. As an example of the latter, if cell i 's architecture chromosome equals, say, 000110000100 then it is connected to cells $i \pm 4$ and $i \pm 7 \pmod{N}$, since 000110 and 000100 are the Gray encodings of the decimal values 4 and 7, respectively.

The algorithm now proceeds as in Section 3; initial configurations are presented and fitness scores of each cell are accumulated over C configurations, after which evolution occurs. As with the original algorithm, a cell has access only to its neighbors and applies genetic operators to the genomes of the fitter ones. Each cell has four connections (in addition to a self-connection), but these need not be identical for all cells, thereby entailing heterogeneous connectivity. We have found that performance can be increased by using slower evolutionary rates for connections than

⁵ A prime characteristic of the Gray code is the adjacency property, i.e., adjacent integers differ by a single bit. This is desirable where genetic operators are concerned [8].

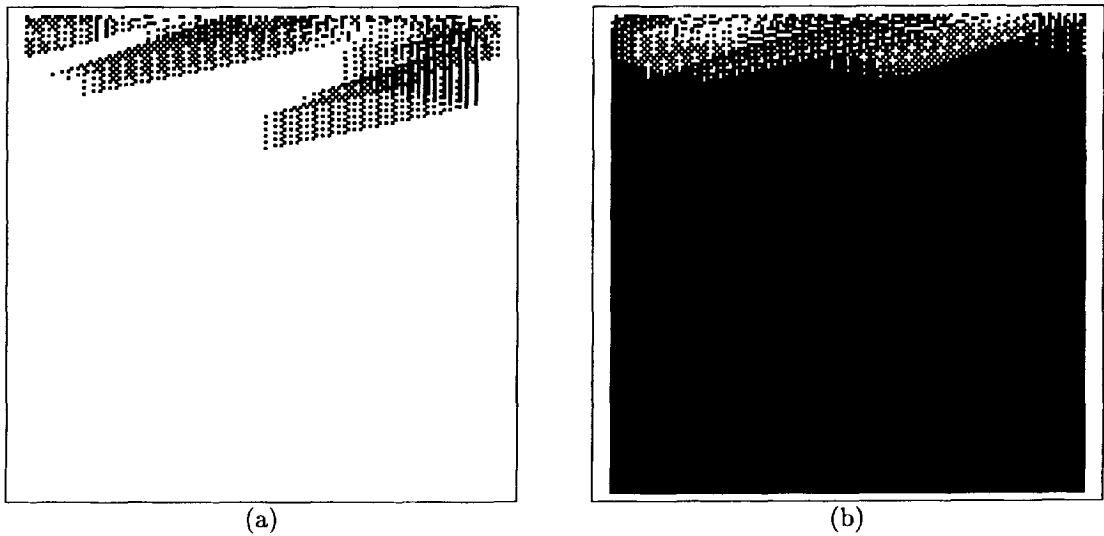


Fig. 10. The density task: operation of a co-evolved, non-uniform, $C_{149}(3, 5)$ CA. (a) Initial density of 1s is 0.48. (b) Initial density of 1s is 0.51. Note that computation time, i.e., the number of time steps until convergence to the correct final pattern, is shorter than that of the GKL rule. Furthermore, it can be qualitatively observed that the computational “behavior” is different than GKL, as is to be expected due to the different connectivity architecture.

for rules. Thus, while rules evolve every $C = 300$ configurations, connections evolve every $C' = 1500$ configurations. The two-level dynamics engendered by the concomitant evolution of rules and connections markedly increases the size of the space searched by evolution. Our results demonstrate that high performance can be attained, nonetheless.

We performed several evolutionary runs using $C_N(1, d_i)$ architectures, two typical results of which are depicted in Fig. 11. We find it quite remarkable that the architectures evolved succeed in “selecting” connection lengths d_i that coincide in most cases with minima points of the acd graph, reflecting the strong correlation between performance and acd. This, along with the high levels of performance attained, demonstrates that evolution has succeeded in finding non-uniform CAs with efficient architectures, as well as rules. In fact, the performance attained is higher than that of the fixed-architecture CAs of Section 4. Fig. 12 demonstrates the operation of a co-evolved, $C_{129}(1, d_i)$ CA on the density task.

As noted in Section 4, Mitchell et al. discussed two possible choices of initial configurations, either uniformly distributed over densities in the range

[0.0, 1.0], or binomially distributed over initial densities. As explained therein, this distinction did not prove essential in our studies and we concentrated on the former distribution; nonetheless, we find that our evolved CAs attain high performance even when applying the binomial distribution. Observing the results presented in Table 2, we note that performance exceeds that of previously evolved CAs, coupled with markedly shorter computation times (as demonstrated, e.g., by Fig. 12). It is important to note that this is achieved using only five connections per cell, as compared to seven used by the fixed, standard-architecture CAs. It is most likely that our CAs could attain even better results using a higher number of connections per cell, since this entails a notable reduction in acd.

6. Co-evolving low cost architectures

In Section 5 we showed that high performance architectures can be co-evolved using the cellular programming algorithm, thus obviating the need to specify in advance the precise connectivity scheme. The mean d_i value of evolved, $C_{129}(1, d_i)$ architectures

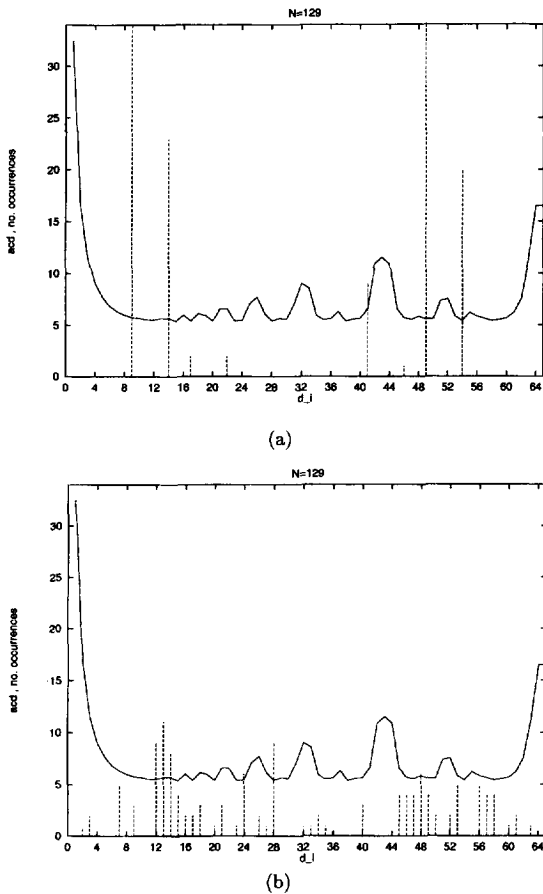


Fig. 11. Evolving architectures. Results of two typical evolutionary runs using $C_{129}(1, d_i)$. Each figure depicts a histogram of the number of occurrences of evolved d_i values for all grid cells, overlaid on the acd graph. Performance in both cases is 0.98. Mean d_i value is 31.5 for run (a), 30.8 for run (b).

was in the range [30, 40] (e.g., Fig. 11). It is natural to ask whether high performance architectures can be evolved, which are also of low *connectivity cost* per cell, defined as d_i for the $C_N(1, d_i)$ case and $a_i + b_i$ for $C_N(a_i, b_i)$.

In order to evolve low cost architectures we employ the cellular programming algorithm of Section 5 with a modified cellular fitness value, f'_i , incorporating the performance of cell i as well as its connectivity cost:

$$f'_i = f_i - \alpha(a_i + b_i)/N$$

for $C_N(a_i, b_i)$ architectures and

$$f'_i = f_i - \alpha d_i/N$$

for $C_N(1, d_i)$ ones, where f_i denotes the original fitness value of cell i as defined in Section 3, and α is a coefficient in the range [0.02, 0.04]. The algorithm now proceeds as in Section 5, with an added evolutionary “pressure” toward low cost architectures.

Fig. 13 depicts the results of two typical evolutionary runs using $C_N(1, d_i)$ architectures. Comparing this figure with Fig. 11, we note that low cost architectures are indeed evolved, exhibiting markedly lower connectivity cost, with only a slight degradation in performance.

In Section 2 we observed that every $C_N(a, b)$ architecture is isomorphic to a $C_N(1, d')$ architecture, for some equivalent d' . We noted that general $C_N(a, b)$ architectures come into play when one wishes to minimize connectivity cost, as well as to maximize performance; the equivalent d' value of a $C_N(a, b)$ architecture may be large, resulting in a lower cost of $C_N(a, b)$ as compared with the isomorphic $C_N(1, d')$ architecture. These observations motivated the evolution of general $C_N(a_i, b_i)$ architectures, a typical result of which is demonstrated in Fig. 14; co-evolved, $C_N(a_i, b_i)$ architectures surpass $C_N(1, d_i)$ ones in that better performance is attainable with considerably lower connectivity cost.

7. Discussion

In this paper we have studied the relationship between performance and connectivity in evolving, non-uniform CAs. Our main findings are:

- (1) The performance of fixed-architecture CAs solving global tasks depends strongly and linearly on their average cellular distance. Compared with the standard $C_N(1, 2)$ architecture, considerably higher performance can be attained at very low connectivity values, by selecting a $C_N(1, d)$ or $C_N(a, b)$ architecture with a low acd value, such that $d, a, b \ll N$.
- (2) High performance architectures can be co-evolved using the cellular programming algorithm, thus obviating the need to specify in advance the precise connectivity scheme. Furthermore, it is possible to evolve such architectures that

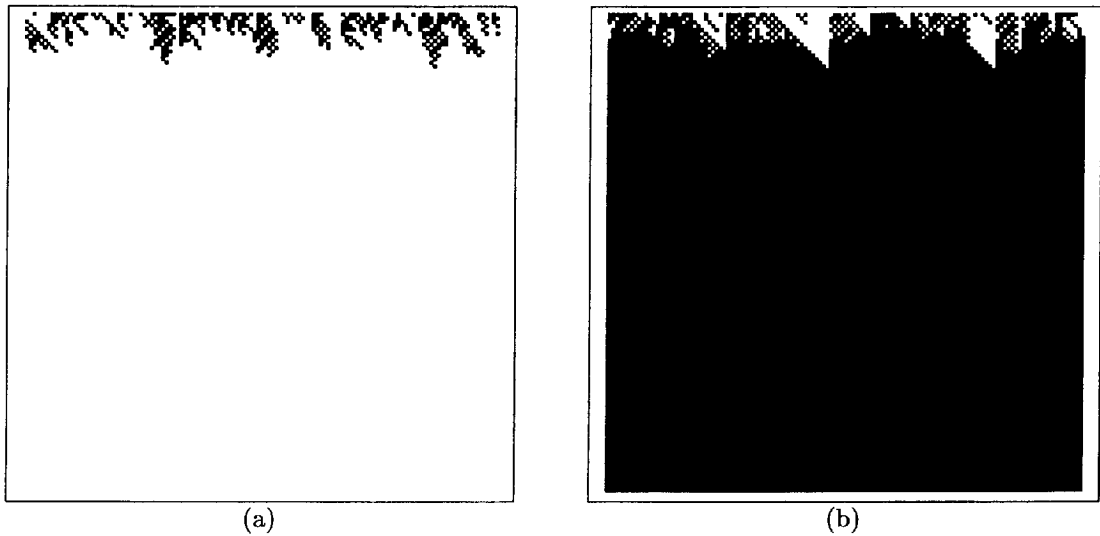


Fig. 12. The density task: operation of a co-evolved, non-uniform, $C_{129}(1, d_i)$ CA. (a) Initial density of 1s is 0.496. (b) Initial density of 1s is 0.504. Note that computation time is shorter than that of the fixed-architecture CA and markedly shorter than the GKL rule.

Table 2
A comparison of performance and computation times of the best CAs

Designation	Rule(s)	Architecture	Connections per cell	$\mathcal{P}_{129,10^4}$	$\mathcal{T}_{129,10^4}$
Co-evolved CA (1)	Evolved, non-uniform	Evolved, non-standard	5	0.791	17
Co-evolved CA (2)	Evolved, non-uniform	Evolved, non-standard	5	0.788	27
Co-evolved CA (3)	Evolved, non-uniform	Evolved, non-standard	5	0.781	12
ϕ_{100}	Evolved, uniform	Fixed, standard	7	0.775	72
ϕ_{11102}	Evolved, uniform	Fixed, standard	7	0.751	80
ϕ_{17083}	Evolved, uniform	Fixed, standard	7	0.743	107
GKL	Designed, uniform	Fixed, standard	7	0.825	74

$\mathcal{P}_{129,10^4}$ is a measure introduced by Mitchell et al., representing the fraction of correct classifications performed by the CA of size $N = 129$ over 10^4 initial configurations randomly chosen from a binomial distribution over initial densities. $\mathcal{T}_{129,10^4}$ denotes the average computation time over the 10^4 initial configurations, i.e., the average number of time steps until convergence to the final pattern. The rules designated by ϕ_i are those reported by Mitchell et al. Co-evolved CA (1) is fully specified in Table 3 of Appendix B.

exhibit low connectivity cost as well as high performance.

We observed that the average cellular distance landscape is rugged and showed that the performance landscape is qualitatively similar. This suggests an added benefit of evolving, heterogeneous architectures over homogeneous, fixed ones: While the latter may get stuck in a low performance local minimum, the evolving architectures, where each cell “selects” its own connectivity, result in a melange of local minima, yielding in many cases higher performance.

We have provided empirical evidence as to the added efficiency of $C_N(1, \sqrt{N})$ architectures in solving global tasks, suggesting that the density problem has a good embedding in two dimensions. A theoretical result by [2] states that the minimal diameter of $C_N(a, b)$ circulant is achieved with $C_N(O(\sqrt{N}), O(\sqrt{N}))$. This suggests that the performance landscape has a global maximum at $a, b = O(\sqrt{N})$ (but with $a \neq b$).

We note in passing that as it is physically possible to construct systems of (up to) three dimensions, one can gain the equivalent of long-range connections

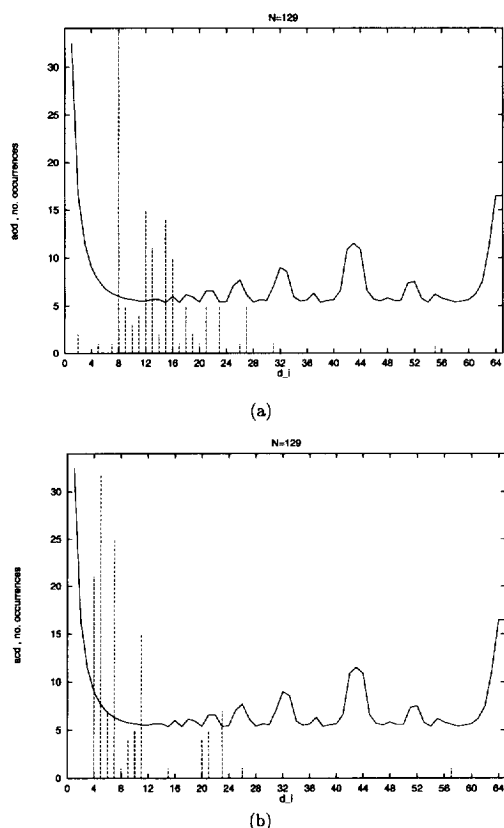


Fig. 13. Evolving low cost architectures. Results of two typical evolutionary runs using $C_{129}(1, d_i)$. Each figure depicts a histogram of the number of occurrences of evolved d_i values for all grid cells, overlaid on the acd graph. (a) Performance is 0.97, mean d_i value is 13.6. (b) Performance is 0.96, mean d_i value is 9.

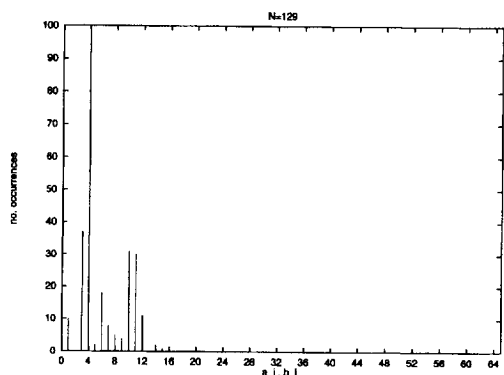


Fig. 14. Evolving low cost architectures. Result of a typical evolutionary run using $C_{129}(a_i, b_i)$. The figure depicts a histogram of the number of occurrences of evolved a_i and b_i values for all grid cells. Performance is 0.97, mean $a_i + b_i$ value is 6.1.

gratuitously; by this we mean that a physical realization of a locally connected, three-dimensional system implicitly “contains” a remotely connected system of lower dimensionality⁶. An interesting extension of our work would be the evolution of architectures using such higher-dimensionality grids, which may result in yet better performance coupled with reduced connectivity cost.

Using our algorithm to solve the density task offers a promising approach for solving a general wiring problem for a set of distributed processors: In this problem one is given a set of processors that should be connected to each other in a way that minimizes average processor distance (i.e., the number of processors a message must traverse on its path between two given processors). Problem constraints may include minimal and maximal connection lengths, pre-specified neighbors for some or all cells, and the (possibly distinct) number of impinging connections per processor. Using our algorithm to solve the density task, where each processor is identified with a cell and connectivity constraints are applied by holding the corresponding connections fixed, will enable the evolution of an efficient wiring scheme for a given distributed computing network, by maximizing the efficiency of global information propagation.

Our simulations have shown that the cellular programming algorithm may degenerate connections. For example, some runs of the short-lines task with evolving $C_N(1, d_i)$ architectures ended up with most cells having $d_i = 0$. This motivates the use of an algorithm starting with a large number of connections per cell, that are reduced by evolution, thus yielding increased performance and lower connectivity cost. Ultimately, we wish to attain a system that can adapt to the problem’s inherent “landscape”.

Evolving, non-uniform CAs hold potential for studying phenomena of interest in areas such as complex systems, artificial life and parallel computation.

⁶ As noted, a two-dimensional, locally connected system of size N can be embedded in a one-dimensional system with connections of length \sqrt{N} . Similarly, a three-dimensional system can be embedded in a two-dimensional system with connections of length $N^{1/3}$, and in a one-dimensional system with connections of length $N^{2/3}$ and $N^{1/3}$.

Table 3

Cell	Rule	d_i	Cell	Rule	d_i	Cell	Rule	d_i	Cell	Rule	d_i
0	135107FF	59	33	035117F7	56	66	135107F7	44	99	135107F7	59
1	135107FF	44	34	115107F7	56	67	135107F7	44	100	035117F7	40
2	135107F7	63	35	115107F7	8	68	135107F7	44	101	135117F7	8
3	035107FF	40	36	135107FF	8	69	135107F7	8	102	035117F7	40
4	035107FF	40	37	135107FF	56	70	035107F7	8	103	035107F7	40
5	035107F7	15	38	035107FF	56	71	035117FF	52	104	035107F7	56
6	035117F7	40	39	035107F7	48	72	035107FF	11	105	135107F7	56
7	035107F7	56	40	035107F7	8	73	035107FF	59	106	035105FF	56
8	135117F7	56	41	035107FF	44	74	035107FF	59	107	035117F7	56
9	035107F7	63	42	135107FF	59	75	035107F7	55	108	135117F7	56
10	035107F7	63	43	135107FF	43	76	035117FF	56	109	135117F7	56
11	035117F7	52	44	135107F7	63	77	035107FF	40	110	135107F7	56
12	035127FF	11	45	035107FF	59	78	035107F7	44	111	035107FF	56
13	035127FF	59	46	035117F7	43	79	135117F7	15	112	135107F7	56
14	135117F7	8	47	035107FF	43	80	035107F7	15	113	135107F7	56
15	035107F7	11	48	035107FF	40	81	035107F7	59	114	135107FF	52
16	135117F7	11	49	035117F7	56	82	135107F7	40	115	035107F7	43
17	035117F7	43	50	035105FF	56	83	035107F7	63	116	035107FF	43
18	135107FF	4	51	035107F7	56	84	035107F7	4	117	035107F7	43
19	035117FF	4	52	035107FF	63	85	035127FF	56	118	035107FF	56
20	035117F7	4	53	135107FF	52	86	135107F7	56	119	135107F7	56
21	035117F7	59	54	035105FF	4	87	135107F7	8	120	035107F7	40
22	135107F7	12	55	135107FF	56	88	035157F7	7	121	135107FF	8
23	135107F7	40	56	135107FF	56	89	035117F7	63	122	035107FF	8
24	135107F7	59	57	035107F7	4	90	035107F7	40	123	035107FF	56
25	035107F7	55	58	035107FF	4	91	035107F7	56	124	135107F7	56
26	135107F7	40	59	135107FF	11	92	035107F7	56	125	035107F7	56
27	035107F7	56	60	135107F7	11	93	035107FF	4	126	035107FF	56
28	035107FF	56	61	035107F7	59	94	035117F7	56	127	035107F7	11
29	035107FF	56	62	035107FF	56	95	135107F7	12	128	135107FF	59
30	035107FF	39	63	135117F7	56	96	035107FF	56			
31	035107F7	56	64	135117F7	48	97	035117FF	63			
32	035117F7	48	65	035117F7	48	98	035107F7	59			

This work has shed light on the importance of selecting efficient CA architectures, and demonstrated the feasibility of their evolution.

Acknowledgements

We are grateful to Melanie Mitchell for her careful reading of this manuscript and her many helpful suggestions. We thank Yossi Azar, Jason Lohn, and Hezy Yeshurun for helpful discussions.

Appendix A. Computing acd and equivalent d'

Determining the diameter and average cellular distance of a general circulant is a difficult problem [3].

The minimum diameter has been determined for all circulants on N nodes and two connection lengths [2]. Our interest is in the special case of $C_N(a, b)$. We observe that by symmetry we need only consider the paths from node 0 to each other node j , $j = 1, \dots, N - 1$ (provided such a path exists). Thus, we express j as $ax + by \pmod N$, $x, y \in [-N, N]$ [1]. The graphs depicted in Section 2 were computed by considering all possible (a, b) pairs. For each such pair, minimum cellular distances from node 0 to all other nodes were computed by considering all possible x, y pairs. The average of these distances was then taken.

To find the isomorphic $C_N(1, d')$ architecture for a given $C_N(a, b)$ we proceed as follows: Consider the list of nodes in the $C_N(a, b)$ graph: $0, 1, \dots, N - 1$. Now rearrange this list such that nodes originally a

units apart are now adjacent (unless $\gcd(a, N) > 1$, in which case b is taken). The equivalent d' is then the minimal number of unit connections to node b from the head of the list (or a , if $\gcd(a, N) > 1$). For example, $C_7(2, 3)$ nodes are rearranged in the following order: 0, 2, 4, 6, 1, 3, 5, and the equivalent d' value is therefore $d' = 2$ (minimal number of unit connections from node 0 to node 3).

Appendix B. Specification of co-evolved CA (1)

Co-evolved CA (1), whose performance measures are given in Table 2, is fully specified in Table 3. As the architecture in question is non-uniform, $C_{129}(1, d_i)$, this involves 129 rules and d_i values. The 32-bit rule string is shown as eight hexadecimal digits, with neighborhood configurations given in lexicographic order; the first (left-most) bit specifies the state to which neighborhood 00000 is mapped to and so on until the last (right-most) bit specifying the state to which neighborhood 11111 is mapped to. The five neighborhood bits represent the values of cells $i - d_i, i - 1, i, i + 1, i + d_i \pmod{N}$, respectively. Cell 0 is the left-most grid cell.

References

- [1] F.T. Boesch and R. Tindell, Circulants and their connectivities, *J. Graph Theory* 8 (1984) 487–499.
- [2] F.T. Boesch and J.-F. Wang, Reliable circulant networks with minimum transmission delay, *IEEE Trans. Circuits and Systems CAS-32* (12) (1985) 1286–1291.
- [3] F. Buckley and F. Harary, *Distance in Graphs* (Addison-Wesley, Redwood City, CA, 1990).
- [4] J.P. Crutchfield and M. Mitchell, The evolution of emergent computation, *Proceedings of the National Academy of Sciences USA*, 92 (23), 1995.
- [5] R. Das, J.P. Crutchfield, M. Mitchell and J.E. Hanson, Evolving globally synchronized cellular automata, in: *Proc. 6th Int. Conf. on Genetic Algorithms*, ed. L.J. Eshelman (Morgan Kaufmann, San Francisco, CA, 1995) pp. 336–343.
- [6] R. Das, M. Mitchell and J.P. Crutchfield, A genetic algorithm discovers particle-based computation in cellular automata, in: *Parallel Problem Solving from Nature- PPSN III*, Vol. 866 of *Lecture Notes in Computer Science*, eds. Y. Davidor, H.-P. Schwefel and R. Männer (Springer, Berlin, 1994) pp. 344–353.
- [7] P. Gacs, G.L. Kurdyumov and L.A. Levin, One-dimensional uniform arrays that wash out finite islands, *Problemy Peredachi Informatsii* 14 (1978) 92–98.
- [8] D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning* (Addison-Wesley, Reading, MA, 1989).
- [9] P. Gonzaga de Sá and C. Maes, The Gacs–Kurdyumov–Levin automaton revisited, *J. Stat. Phys.* 67 (3/4) (1992) 507–522.
- [10] H. Hartman and G.Y. Vichniac, Inhomogeneous cellular automata, in: *Disordered Systems and Biological Organization*, eds. E. Bienenstock, F. Fogelman and G. Weisbuch (Springer, Berlin, 1986) pp. 53–57.
- [11] S. Haykin, *Digital Communications* (Wiley, New York, 1988).
- [12] M. Land and R.K. Belew, No perfect two-state cellular automata for density classification exists, *Phys. Rev. Lett.* 74 (25) (1995) 5148–5150.
- [13] M. Mitchell, J.P. Crutchfield and P.T. Hraber, Dynamics, computation, and the “edge of chaos”: A re-examination, in: *Complexity: Metaphors, Models and Reality*, eds. G. Cowan, D. Pines and D. Melzner (Addison-Wesley, Reading, MA, 1994) pp. 491–513.
- [14] M. Mitchell, J.P. Crutchfield and P.T. Hraber, Evolving cellular automata to perform computations: Mechanisms and impediments, *Physica D* 75 (1994) 361–391.
- [15] M. Mitchell, P.T. Hraber and J.P. Crutchfield, Revisiting the edge of chaos: Evolving cellular automata to perform computations, *Complex Systems* 7 (1993) 89–130.
- [16] 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, Singapore, 1988) pp. 293–301.
- [17] M. Sipper, Non-uniform cellular automata: Evolution in rule space and formation of complex structures, in: *Artificial Life IV*, eds. R.A. Brooks and P. Maes (MIT Press, Cambridge, MA, 1994) pp. 394–399.
- [18] M. Sipper, Quasi-uniform computation-universal cellular automata, in: *ECAL'95: 3rd European Conf. on Artificial Life*, eds. F. Morán, A. Moreno, J.J. Merelo and P. Chacón, *Lecture Notes in Computer Science*, Vol. 929 (Springer, Berlin, 1995) pp. 544–554.
- [19] M. Sipper, Studying artificial life using a simple, general cellular model, *Artificial Life J.* 2 (1) (1995) 1–35.
- [20] M. Sipper, Co-evolving non-uniform cellular automata to perform computations, *Physica D* 92 (1996) 193–208.
- [21] T. Toffoli and N. Margolus, *Cellular Automata Machines* (MIT Press, Cambridge, MA, 1987).
- [22] G.Y. Vichniac, P. Tamayo and H. Hartman, Annealed and quenched inhomogeneous cellular automata, *J. Stat. Phys.* 45 (1986) 875–883.
- [23] S. Wolfram, Universality and complexity in cellular automata, *Physica D* 10 (1984) 1–35.