Co-evolving Cellular Architectures by Cellular Programming

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

Eytan Ruppin
Department of Computer Science
Tel Aviv University
Tel Aviv 69978, Israel
ruppin@math.tau.ac.il

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

I. 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. CAs exhibit three notable features, namely massive parallelism, locality of cellular interactions, and simplicity of basic components (cells). A major impediment preventing ubiquitous computing with CAs stems from the difficulty of utilizing their complex behavior to perform useful computations. 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 [11].

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 2-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 (Figure 1).

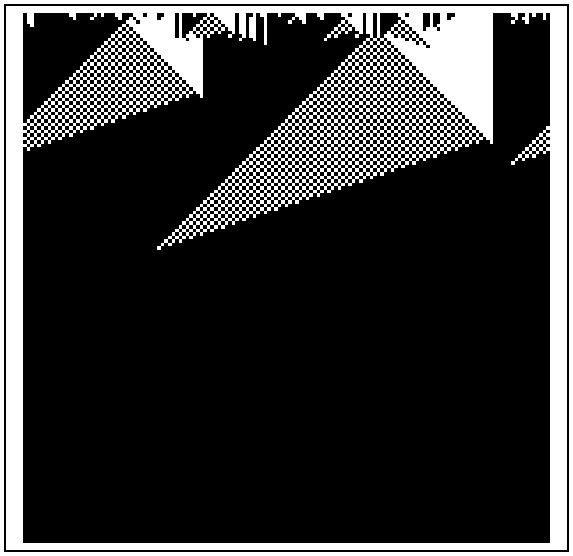

Fig. 1. The density task: Operation of the GKL rule. CA is one-dimensional, uniform, 2-state, with a standard architecture of connectivity radius $r = 3$. Grid size is $N = 149$. White squares represent cells in state 0, black squares represent cells in state 1. The pattern of configurations is shown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

The density task was studied by [12; 11; 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 [14; 16; 15]. Employing this model we found that high performance can be attained for the density task by means of co-evolution [17].

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 [12; 11; 10; 4; 6; 5; 13]. Note that the density task cannot be perfectly solved by a uniform, two-state CA, as recently proven by [9]; however, no upper bound is currently available on the best possible imperfect performance, attained to date by the Gacs-Kurdyumov-Levin (GKL) rule [7; 8] (Figure 1).

Previous studies of the density task were conducted us-

ing locally-connected, one-dimensional grids [11; 17]. 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 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 next Section) as the prime architectural parameter which linearly determines CA performance. Furthermore, we find that high performance architectures can be co-evolved, concomitantly with the rules. This work extends our previous work on the co-evolution of non-uniform CAs [17] 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 the next section we describe the CA architectures studied in this work. In Section III we present the cellular programming algorithm used to co-evolve non-uniform CAs. Section IV discusses CA rule evolution with fixed architectures. In Section V we extend our evolutionary algorithm such that the architecture evolves as well as the cellular rules. Our findings and their possible future application to designing distributed computer networks are discussed in Section VI.

II. 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 (Figure 2). This type of architecture belongs to the general class of *circulant graphs* [3]: For a given positive integer N, let $n_1, n_2, \ldots, n_k$ be a sequence of integers where $0 < n_1 < n_2 < \cdots < n_k < (N+1)/2$. Then the *circulant graph* $C_N(n_1, n_2, \ldots, n_k)$ is the graph on N nodes $v_1, v_2, \ldots, v_N$ 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 $C_N(a, b)$.

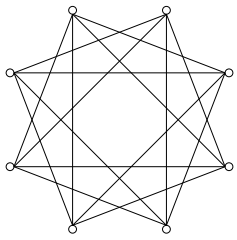

Fig. 2. A $C_8(2,3)$ circulant graph. Each node is connected to four neighbors, with connection lengths of 2 and 3.

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*). The *acd landscape*, obtained by plotting the *acd* of $C_N(a, b)$ architectures as a function of (a, b), is extremely rugged. This is due to the relationship between a and b - if $\gcd(a, b) \neq 1$ the *acd* is markedly higher than when $\gcd(a, b) = 1$ (note that the circulant graph $C_N(n_1, n_2, \ldots, n_k)$ is connected if and only if $\gcd(n_1, n_2, \ldots, n_k, N) = 1$ [1]).

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'. 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 $C_N(1, d')$[1]. We may therefore study the performance of $C_N(1, d)$ architectures, our

[1]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.

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 $C_{101}(1,32)$).

Figure 3 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* declines and reaches a minimum at $d = O(\sqrt{N})$. This supports the notion put forward in Section I concerning the advantage of two-dimensional grids.

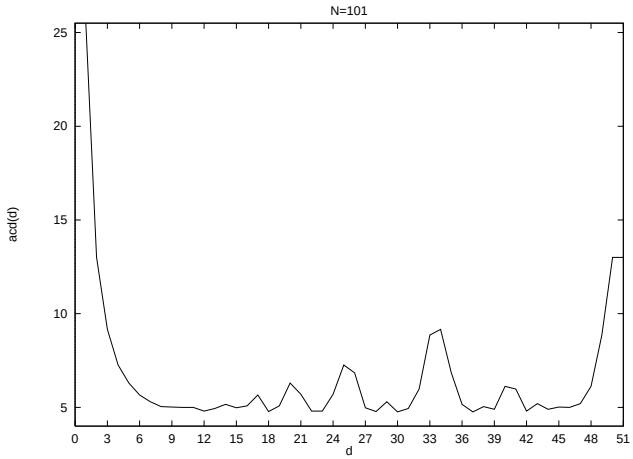

Fig. 3. $C_{101}(1,d)$: Average cellular distance (*acd*) as a function of d. Each d value entails a different $C_{101}(1,d)$ architecture, whose *acd* is represented as a point in the graph. *acd* is plotted for $d \leq N/2$, as it is symmetric about $d = N/2$.

We concentrate on the following issues:
1. How strongly does the *acd* 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?

III. The cellular programming algorithm

We study 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

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

Fig. 4. Cellular programming pseudo-code.

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 Figure 4. In our simulations, the total number of initial configurations per evolutionary run was in the range $[50000, 500000]$[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.

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 con-

[2]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 $[10^6, 3 \cdot 10^6]$ configurations per evolutionary run.

trast, 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.

IV. 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 $[0.0, 1.0]$. Figure 5 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, we have found that it is *linearly* correlated with *acd* (with a correlation coefficient of 0.99, and a negligible p value).

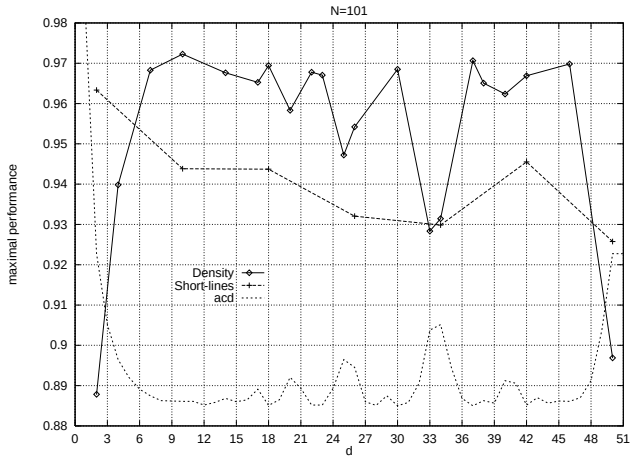

Fig. 5. $C_{101}(1,d)$: Maximal evolved performance on the density and short-lines tasks as a function of d. The graph represents the average results of 420 evolutionary runs; 21 d values were tested for the density task and 7 for the short-lines task. For each such d value, 15 evolutionary runs were 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).

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 random length between $1-7$, the CA must reach a final configuration in which the intervals form continuous lines. 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). Figure 5 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*. As an example of such an architecture, the operation of a co-evolved, $C_{149}(3,5)$ CA on the density task is demonstrated in Figure 6.

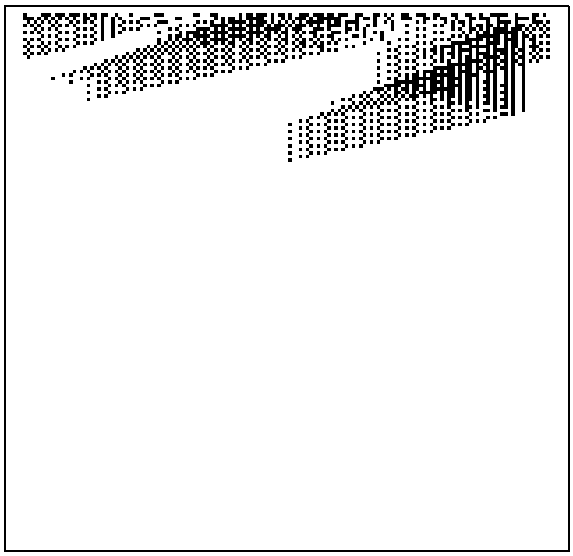

Fig. 6. The density task: Operation of a co-evolved, non-uniform, $C_{149}(3,5)$ CA. Initial density of 1s is 0.48. Note that computation time, i.e., the number of time steps until convergence to the correct final pattern, is shorter than that of the standard-architecture CA depicted in Figure 1. Furthermore, it can be qualitatively observed that the computational "behavior" is different, as is to be expected due to the different connectivity architectures.

V. Evolving architectures

In the previous section 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 cellular programming algorithm of Section III is modified, such that each cellular "genome" consists of two "chromosomes". The first, encoding the rule table, is identical to that delineated in Section III, while the second chromosome encodes the cell's connections. 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, a typical result of which is depicted in Figure 7. 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 IV. Figure 8 demonstrates the operation of a co-evolved, $C_{129}(1, d_i)$ CA on the density task.

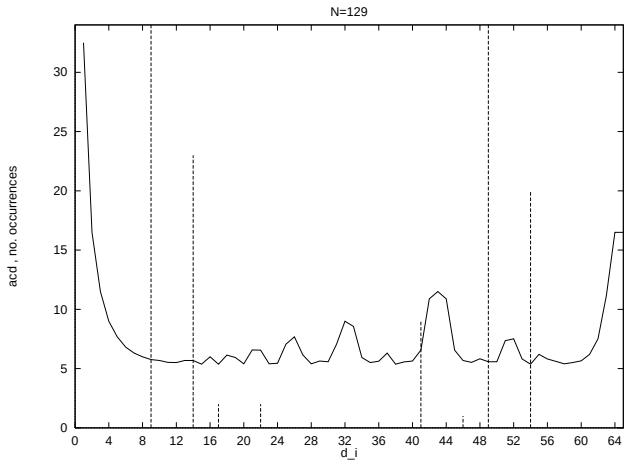

Fig. 7. Evolving architectures. Result of a typical evolutionary run using $C_{129}(1, d_i)$. The figure depicts a histogram of the number of occurrences of evolved d_i values for all grid cells, overlaid on the *acd* graph. Performance is 0.98.

VI. 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.

Fig. 8. The density task: Operation of a co-evolved, non-uniform, $C_{129}(1, d_i)$ CA. 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 standard-architecture CA.

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.

In Section V we showed that high performance architectures can be co-evolved; a possible extension is the evolution of such architectures, which also exhibit low *connectivity cost* per cell, defined as d_i for the $C_N(1, d_i)$ case and $a_i + b_i$ for $C_N(a_i, b_i)$. This may be achieved, e.g., by employing the cellular programming algorithm of Section V using a modified cellular fitness function, incorporating the performance of a cell as well as its connectivity cost. The study of low cost architectures is planned in the future.

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 $C_N(a, b)$ circulants 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 $a \neq b$).

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

Acknowledgments

We are grateful to Yossi Azar, Jason Lohn, Melanie Mitchell, and Hezy Yeshurun for helpful discussions.

References

[1] F. T. Boesch and R. Tindell. Circulants and their connectivities. *Journal of Graph Theory*, 8:487–499, 1984.

[2] F. T. Boesch and J. -F. Wang. Reliable circulant networks with minimum transmission delay. *IEEE Transactions on Circuits and Systems*, CAS-32(12):1286–1291, December 1985.

[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 L. J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 336–343, San Francisco, CA, 1995. Morgan Kaufmann.

[6] R. Das, M. Mitchell, and J. P. Crutchfield. A genetic algorithm discovers particle-based computation in cellular automata. In Y. Davidor, H. -P. Schwefel, and R. Männer, editors, *Parallel Problem Solving from Nature- PPSN III*, volume 866 of *Lecture Notes in Computer Science*, pages 344–353, Berlin, 1994. Springer-Verlag.

[7] P. Gacs, G. L. Kurdyumov, and L. A. Levin. One-dimensional uniform arrays that wash out finite islands. *Problemy Peredachi Informatsii*, 14:92–98, 1978.

[8] P. Gonzaga de Sá and C. Maes. The Gacs-Kurdyumov-Levin automaton revisited. *Journal of Statistical Physics*, 67(3/4):507–522, 1992.

[9] M. Land and R. K. Belew. No perfect two-state cellular automata for density classification exists. *Physical Review Letters*, 74(25):5148–5150, June 1995.

[10] M. Mitchell, J. P. Crutchfield, and P. T. Hraber. Dynamics, computation, and the "edge of chaos": A re-examination. In G. Cowan, D. Pines, and D. Melzner, editors, *Complexity: Metaphors, Models and Reality*, pages 491–513. Addison-Wesley, Reading, MA, 1994.

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

[12] M. Mitchell, P. T. Hraber, and J. P. Crutchfield. Revisiting the edge of chaos: Evolving cellular automata to perform computations. *Complex Systems*, 7:89–130, 1993.

[13] N. H. Packard. Adaptation toward the edge of chaos. In J. A. S. Kelso, A. J. Mandell, and M. F. Shlesinger, editors, *Dynamic Patterns in Complex Systems*, pages 293–301. World Scientific, Singapore, 1988.

[14] M. Sipper. Non-uniform cellular automata: Evolution in rule space and formation of complex structures. In R. A. Brooks and P. Maes, editors, *Artificial Life IV*, pages 394–399, Cambridge, Massachusetts, 1994. The MIT Press.

[15] M. Sipper. Quasi-uniform computation-universal cellular automata. In F. Morán, A. Moreno, J. J. Merelo, and P. Chacón, editors, *ECAL'95: Third European Conference on Artificial Life*, volume 929 of *Lecture Notes in Computer Science*, pages 544–554, Berlin, 1995. Springer-Verlag.

[16] M. Sipper. Studying artificial life using a simple, general cellular model. *Artificial Life Journal*, 2(1):1–35, 1995. The MIT Press, Cambridge, MA.

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