

Cellular Programming

Moshe Sipper

www.moshesipper.com

Copyright ©Moshe Sipper

GENESIS

- **Cellular automata (CA)**
 - Massive parallelism.
 - Locality of cellular interactions.
 - Simplicity of basic components (cells).
- **Major problem:** How do we utilize the complex behavior of CAs to perform useful computations?
- **Non-uniform CAs**

Same as uniform model, except for cellular rules that need not be identical for all cells.
- **Cellular Programming**

Co-evolve non-uniform CAs to perform computations.
- **Applications:** Complex systems, artificial life, parallel computation.

Outline

- Previous work.
- Cellular programming.
- Results: Co-evolved, non-uniform CAs performing computational tasks.
- Conclusions, Discussion, The Future.

Previous work

- [Packard, 1988],
[Mitchell *et al.*, 1993–] (EVCA group, SFI)
Evolution of **uniform** cellular automata
using genetic algorithms.
- [Sipper, 1994–]
Non-uniforms CAs
 - Artificial life “organisms” (reproduction, growth, mobility), evolution in various “environments”.
 - Universal Computation.
 - Co-evolution of computation.

The cellular programming algorithm

Goal: Evolve non-uniform CAs to perform computations.

- Each cell maintains a genome, delineating its rule table.
- Evolution is **local**.
 - Cell “sees” only local neighborhood during evolution.
 - No global fitness ranking nor global genetic operators.
- **Co-evolution**.
 - Standard genetic algorithm: Population of independent problem solutions.
 - Our algorithm: Cell’s fitness depends upon its evolving neighbors.

Computational tasks

- Density (one/two dimensional).
- Synchronization (one/two dimensional).
- Ordering (one dimensional).
- Rectangle-boundary (two dimensional).

Computational tasks

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

Tasks are **non-trivial**:

- A global property is computed, using locally interconnected cells.
- Computation involved corresponds to recognition of a non-regular language.

Results

- Minimal cellular spaces are used:
1D CAs: 2-state, $r = 1$.
2D CAs: 2-state, 5-neighbor.
- Density, Ordering: High performance.
- Synchronization, Rectangle-boundary: Near-perfect performance.
- Co-evolved CAs: Quasi-uniform.
- Co-evolved, one-dimensional CAs are better than **any** possible uniform CA.

Density

(one/two dimensional)

The 2-state CA must decide whether or not the initial configuration contains more than 50% 1s. 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.

Synchronization

(one/two dimensional)

Given any initial configuration, the CA must reach a final configuration, within M time steps, that oscillates between all 0s and all 1s on successive time steps.

Ordering

(one dimensional)

Given any initial configuration, the CA must reach a final configuration in which all 0s are placed on the left side of the grid and all 1s on the right side.

Rectangle-boundary

(two dimensional)

Given an initial configuration consisting of a non-filled rectangle, the CA must reach a final configuration in which the rectangular region is filled, i.e., all cells within the confines of the rectangle are in state 1, and all other cells are in state 0.

Initial configurations consist of random-sized rectangles placed randomly on the grid.

Conclusions

- Non-uniform CAs can attain high performance on non-trivial computational tasks.
- Such CAs can be co-evolved.
The resulting, high performance systems are quasi-uniform.
- Non-uniformity may reduce connectivity requirements, i.e., the use of smaller neighborhoods is made possible.

Conclusions (cont'd):

Advantages of non-uniformity

- *Increased search space size.* Rather than impeding the evolutionary process, this engenders new evolutionary paths leading to high performance systems.
- *Increased variability,* thereby entailing easier “adaptation” to a possible change in the “environment”, i.e., task.
- *Fault tolerance* arising from the insensitivity to minor differences between cellular rules.
- *Scalability* of the evolutionary algorithm.

Future work

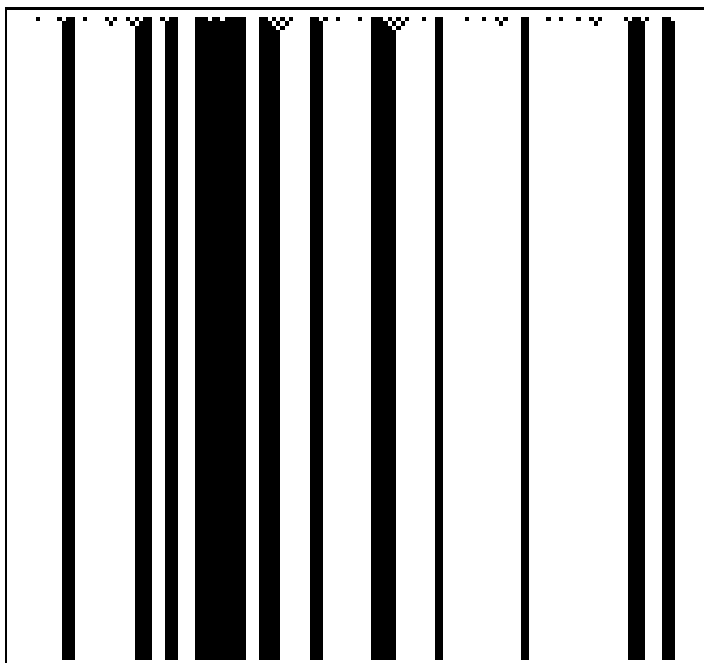
- Expand the range of tasks solvable by cellular programming.
- Co-evolution of architectures.
- Parallel, hardware implementation.
- Understanding how evolution creates complex, global behavior in locally interconnected systems of simple parts.

What else?

- Evolving architectures
- Asynchronous CAs
- Fault-tolerance (non-determinism)
- Evolvable
- Modifications of cellular programming algorithm

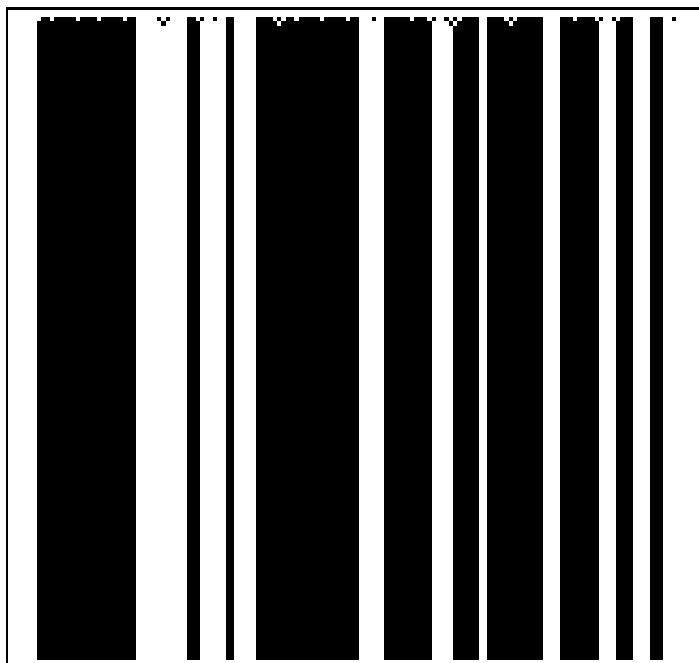
initial density

0.40



0.32

0.60



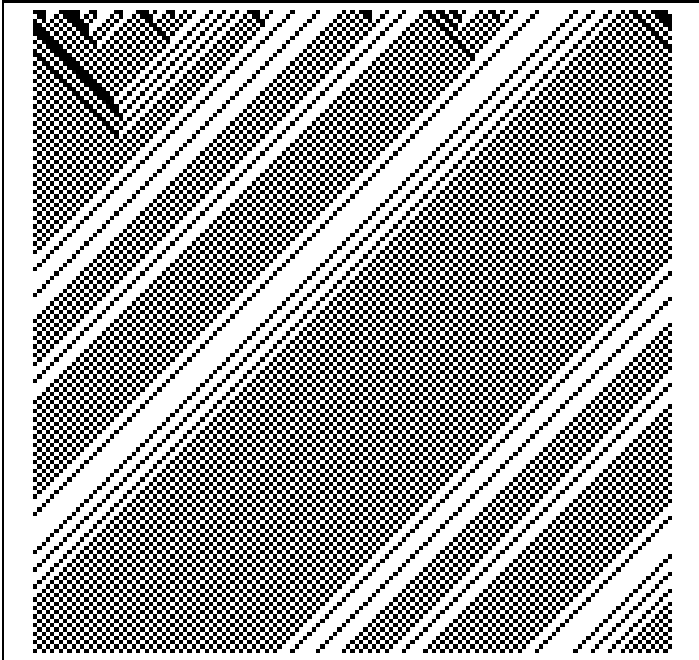
0.66

final density

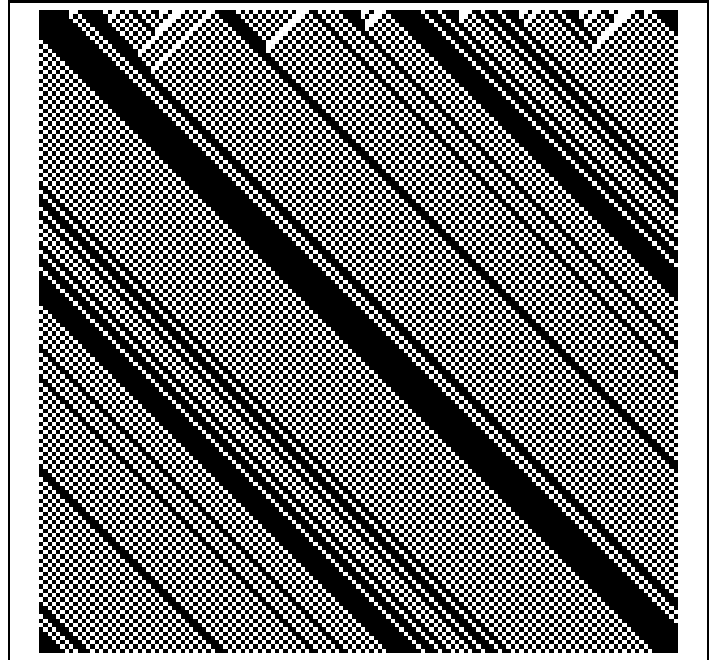
Density task (1D):
Uniform CA, Rule 232 (majority).

initial density

0.40



0.60



0.40

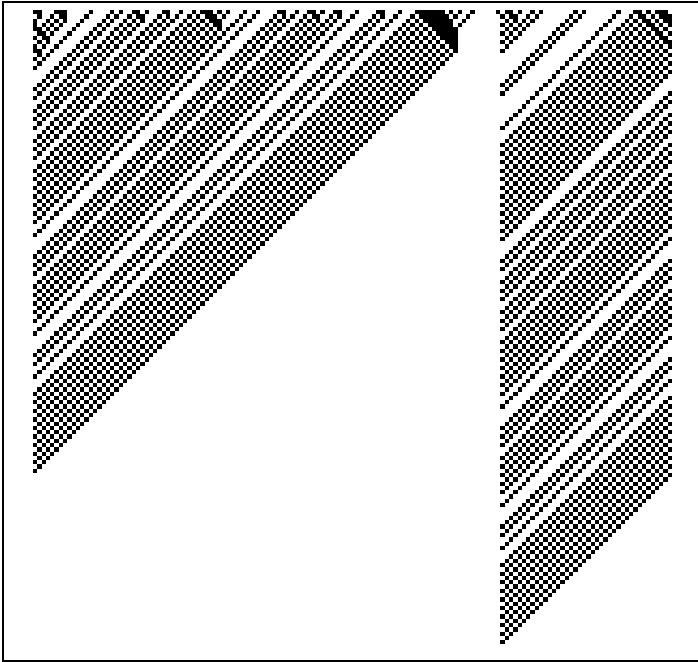
final density

0.60

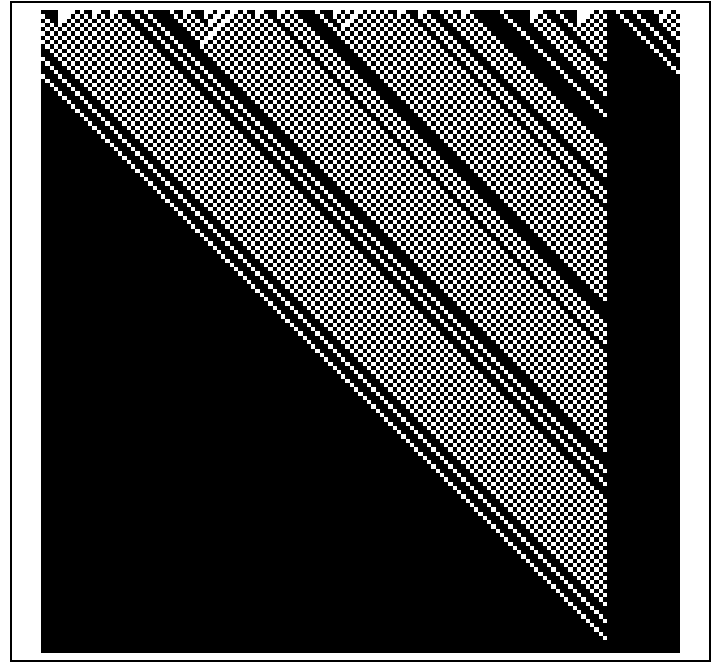
**Density task (1D):
Uniform CA, Rule 226.**

initial density

0.40



0.60



0

1

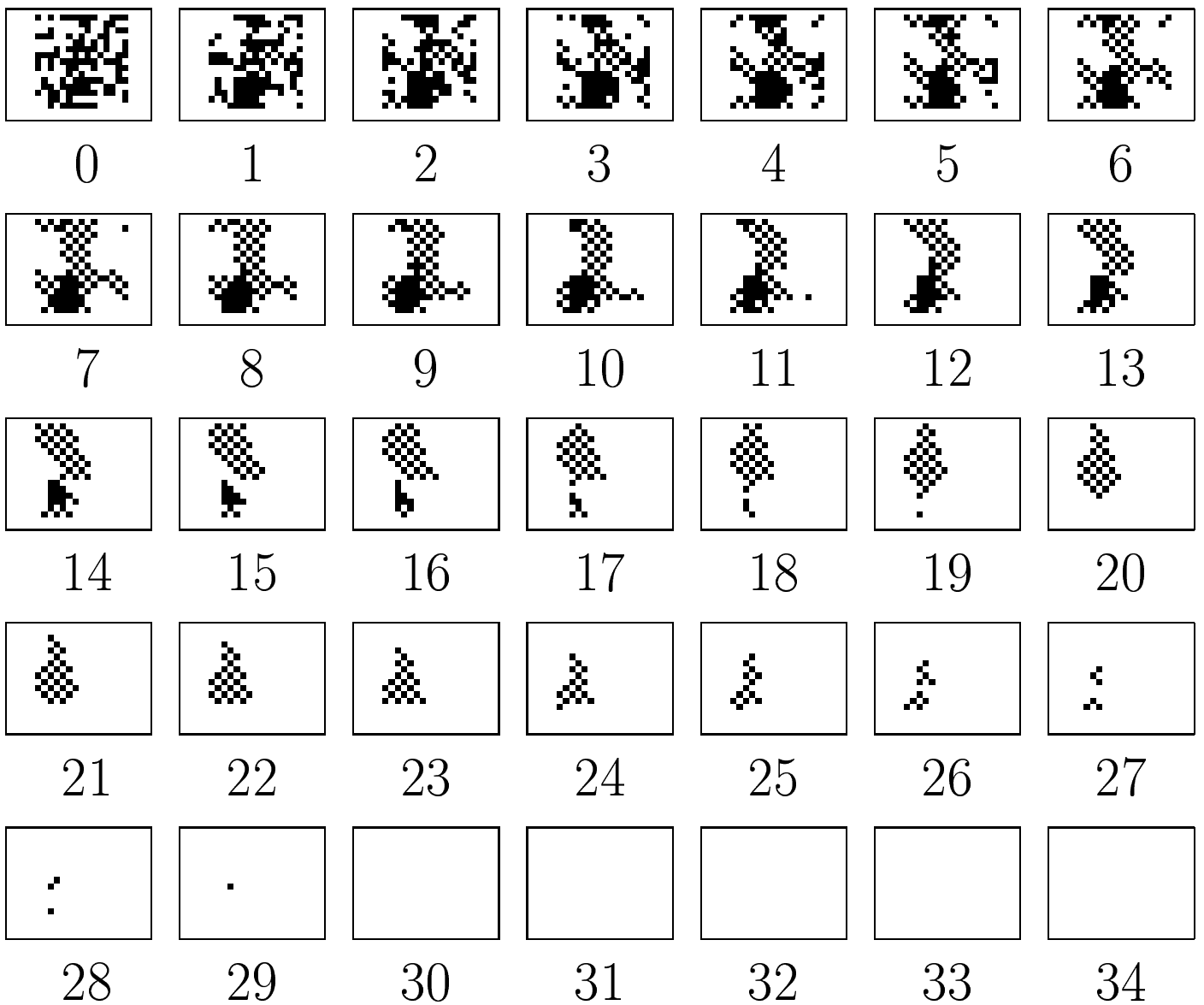
final density



Density task (1D):

A co-evolved, non-uniform CA.

Bottom figures depict rule maps.

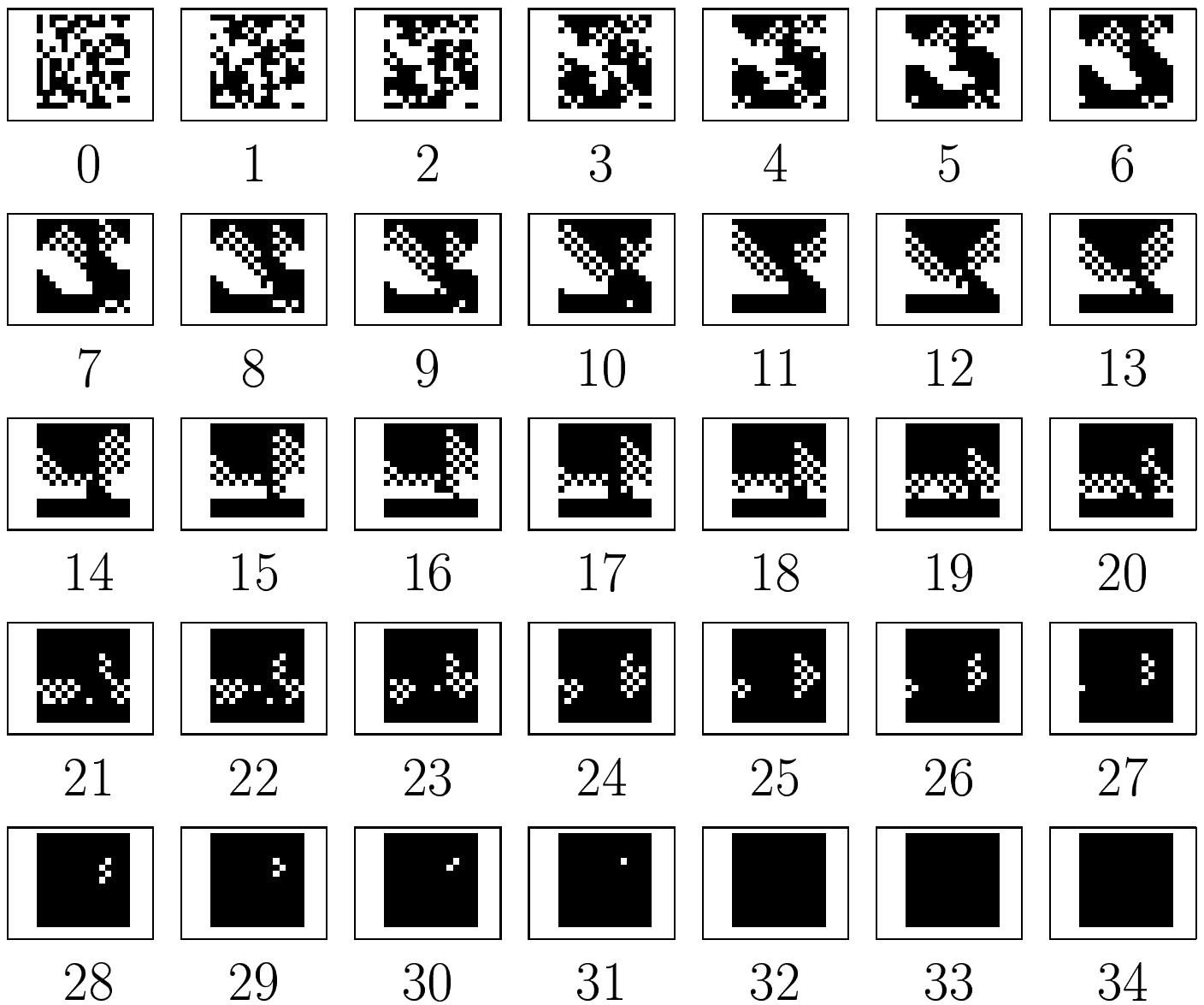


Density task (2D):

A co-evolved, non-uniform CA.

Initial density of 1s is 0.49, final density is 0.

Numbers at bottom of images denote time steps.

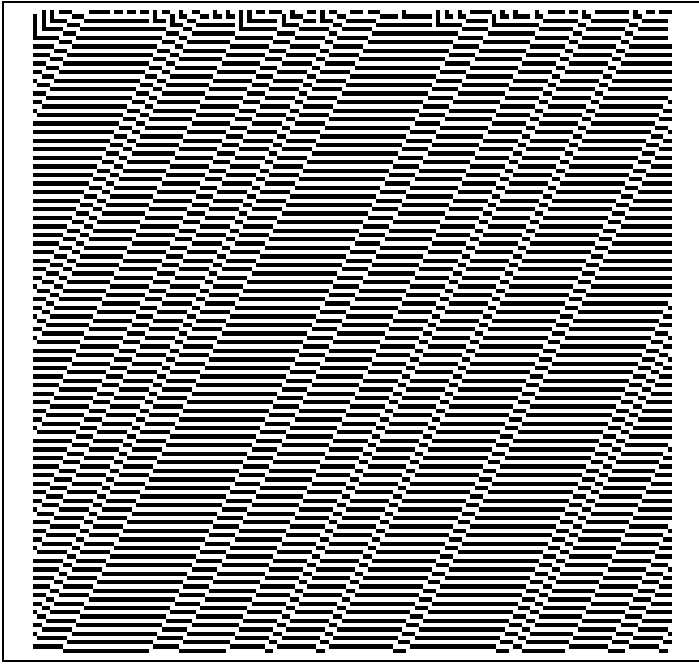


Density task (2D):

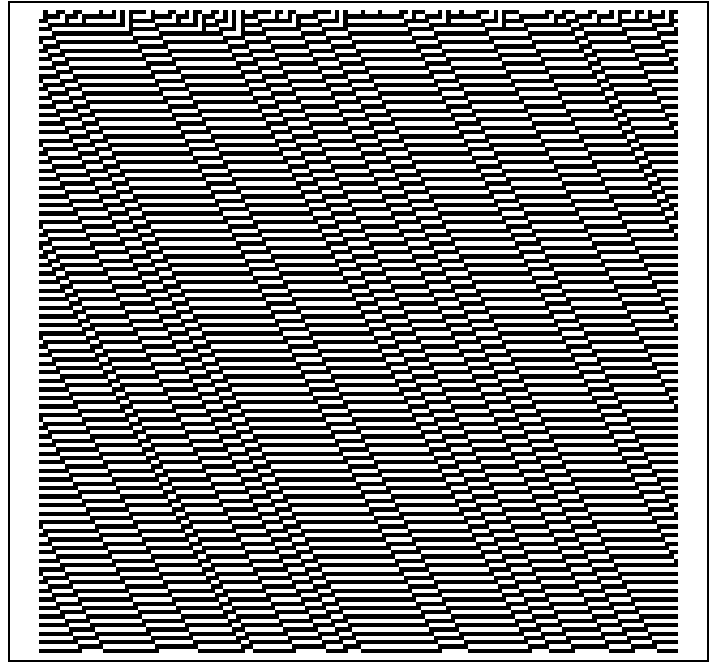
A co-evolved, non-uniform CA.

Initial density of 1s is 0.51, final density is 1.

Numbers at bottom of images denote time steps.

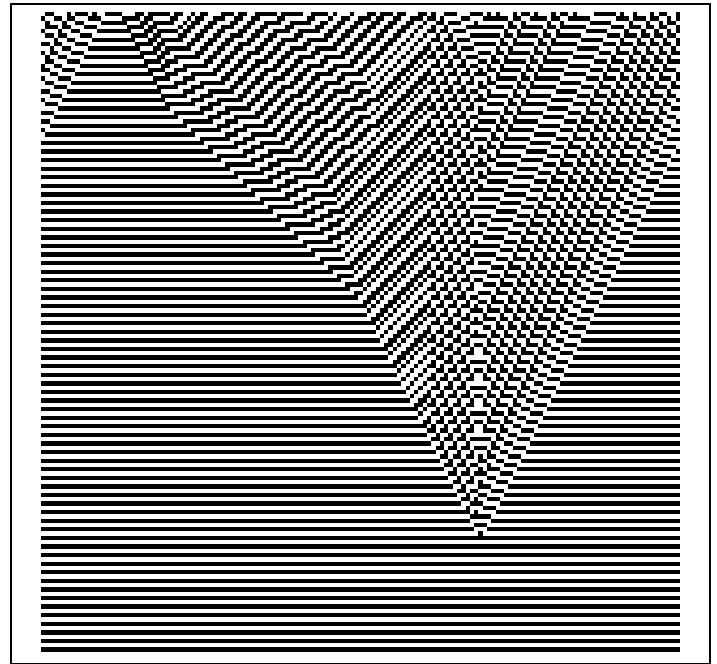
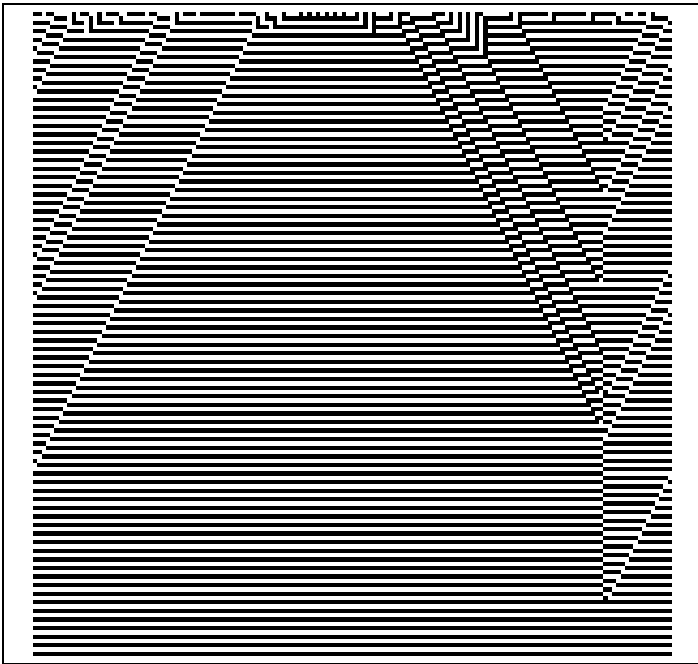


Rule 21

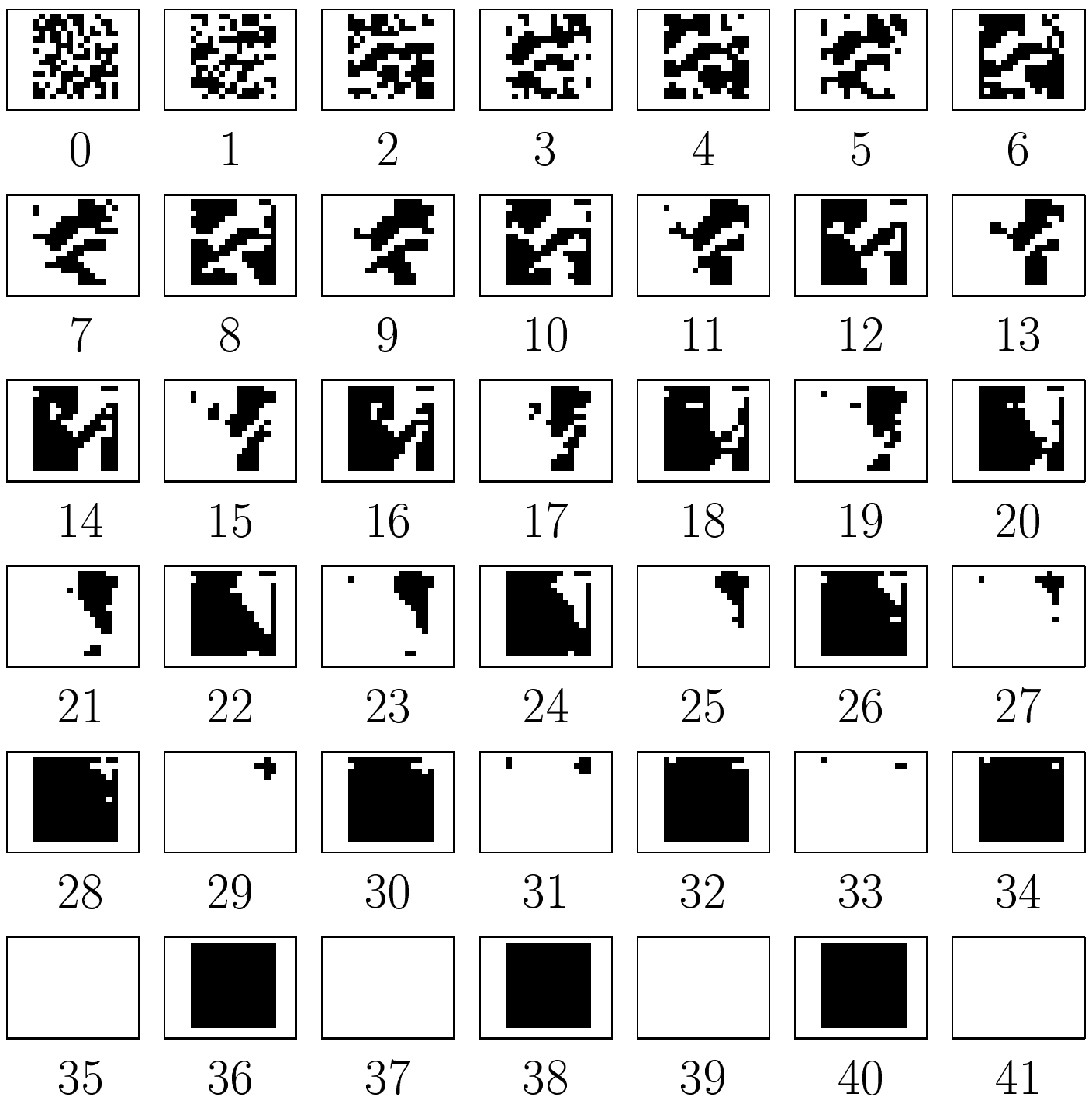


Rule 31

**Synchronization task (1D):
Uniform CAs.**



Synchronization task (1D):
Two co-evolved, non-uniform CAs.
Bottom figures depict rule maps.



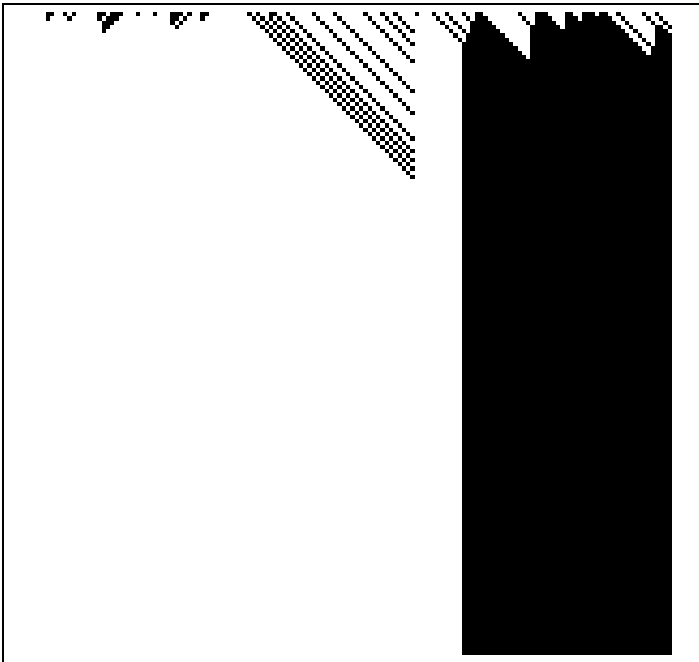
Synchronization task (2D):

A co-evolved, non-uniform CA.

Numbers at bottom of images denote time steps.

initial density

0.315



0.60



0.328

final density

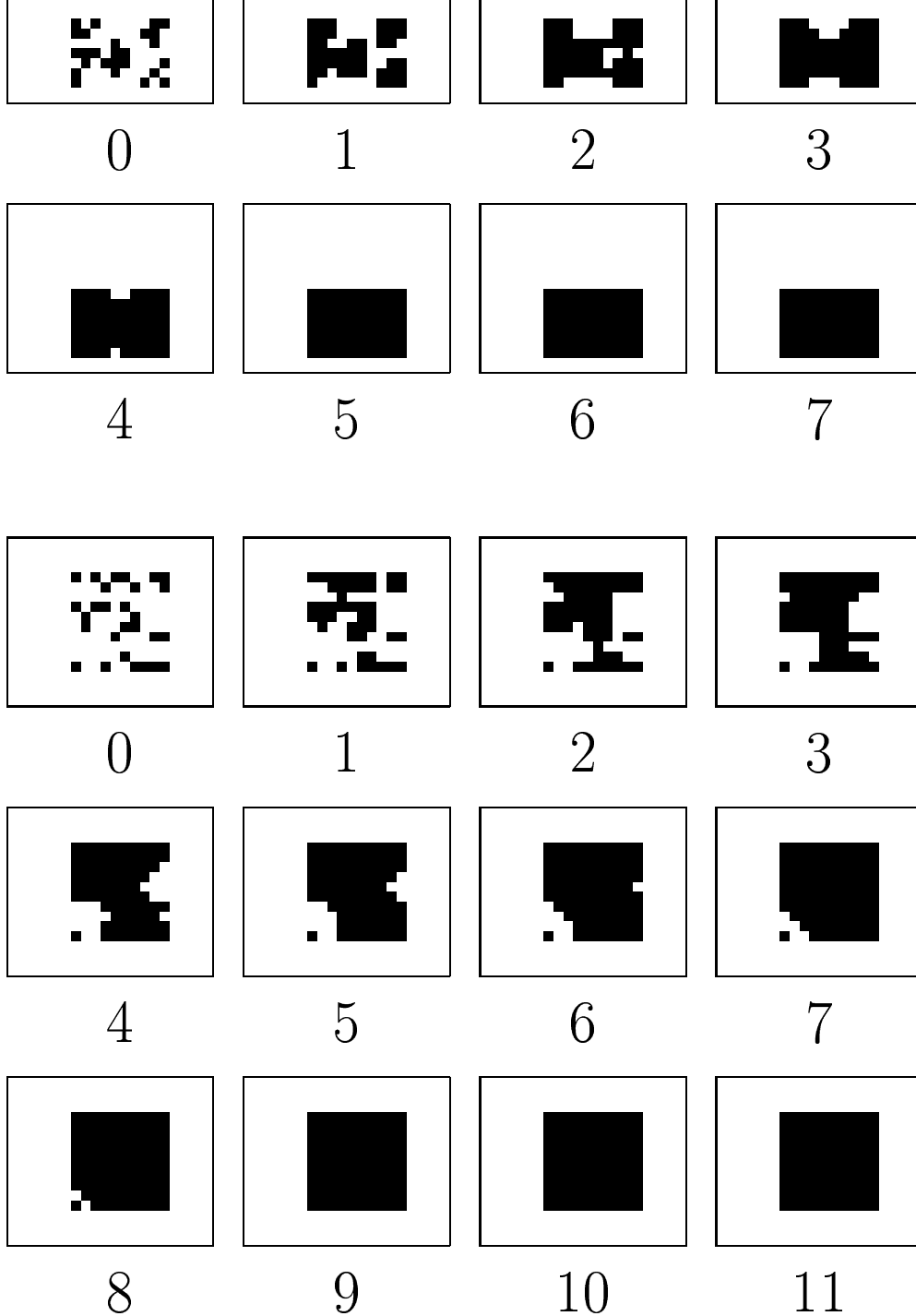
0.59



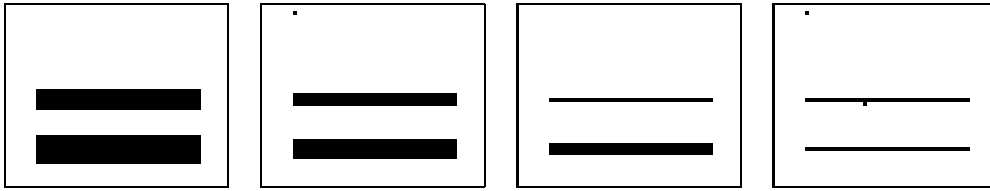
Ordering task (1D):

A co-evolved, non-uniform CA.

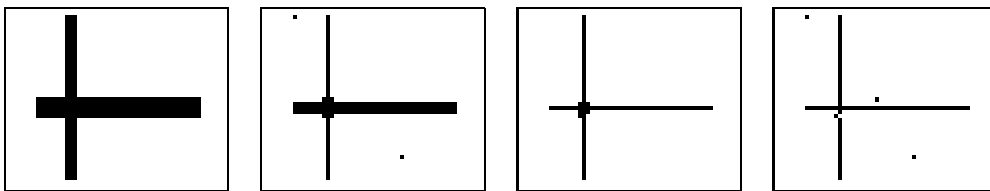
Bottom figures depict rule maps.



Rectangle-boundary task (2D):
A co-evolved, non-uniform CA.
Numbers at bottom of images de-
note time steps.



(a) Two separate lines



(b) Two intersecting lines

Thinning task (2D):

A co-evolved, non-uniform CA.

Numbers at bottom of images denote time steps.

```

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

```

Cellular programming algorithm

One-dimensional, 2-state, $r=1$ CA

[Mitchell, 1996]

Rule table:

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

Grid:

$t = 0$

0	1	1	0	1	0	1	1	0	1	1	0	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

$t = 1$

1	1	1	1	0	1	1	1	1	1	1	0	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

References

- **Cellular Programming:**

- M. Sipper, *Evolution of Parallel Cellular Machines: The Cellular Programming Approach*, Springer-Verlag, Heidelberg, 1997.
- M. Sipper and E. Ruppin, “Co-evolving architectures for cellular machines,” *Physica D*, vol. 99, pp. 428–441, 1997.
- M. Sipper and M. Tomassini, “Computation in artificially evolved, non-uniform cellular automata,” *Theoretical Computer Science*, vol. 217, no. 1, pp. 81–98, March 1999.

- **Cellular Computing:**

- M. Sipper, “The emergence of cellular computing,” *IEEE Computer*, vol. 32, no. 7, pp. 18–26, July 1999.

- **Evolvable Hardware:**

- M. Sipper and D. Mange, Eds., *IEEE Transactions on Evolutionary Computation: Special Issue – From Biology to Hardware and Back*, vol. 3, no. 3, September 1999.
- M. Sipper and E. M. A. Ronald, “A new species of hardware,” *IEEE Spectrum*, vol. 37, no. 3, pp. 59–64, March 2000.