CS 365
Introduction to Scientific Modeling
Lecture 2: Cellular Automata

Stephanie Forrest
Dept. of Computer Science
Univ. of New Mexico
Fall Semester, 2014

THE UNIVERSITY of NEW MEXICO

Reading Assignment

- Mitchell, Ch. 10, 2
- Wolfram, *A New Kind of Science* Ch. 8
- Axelrod: Agent-based modeling as a bridge between disciplines

Review

- Three legs of science
  - Experiment
  - Mathematical theory
  - Simulation and modeling
- What are models good for?
  - Tools for analyzing data
  - Methods for discovering new knowledge (3rd leg)
  - Understanding nature as an information-processing system
  - Explaining how something works---mechanisms

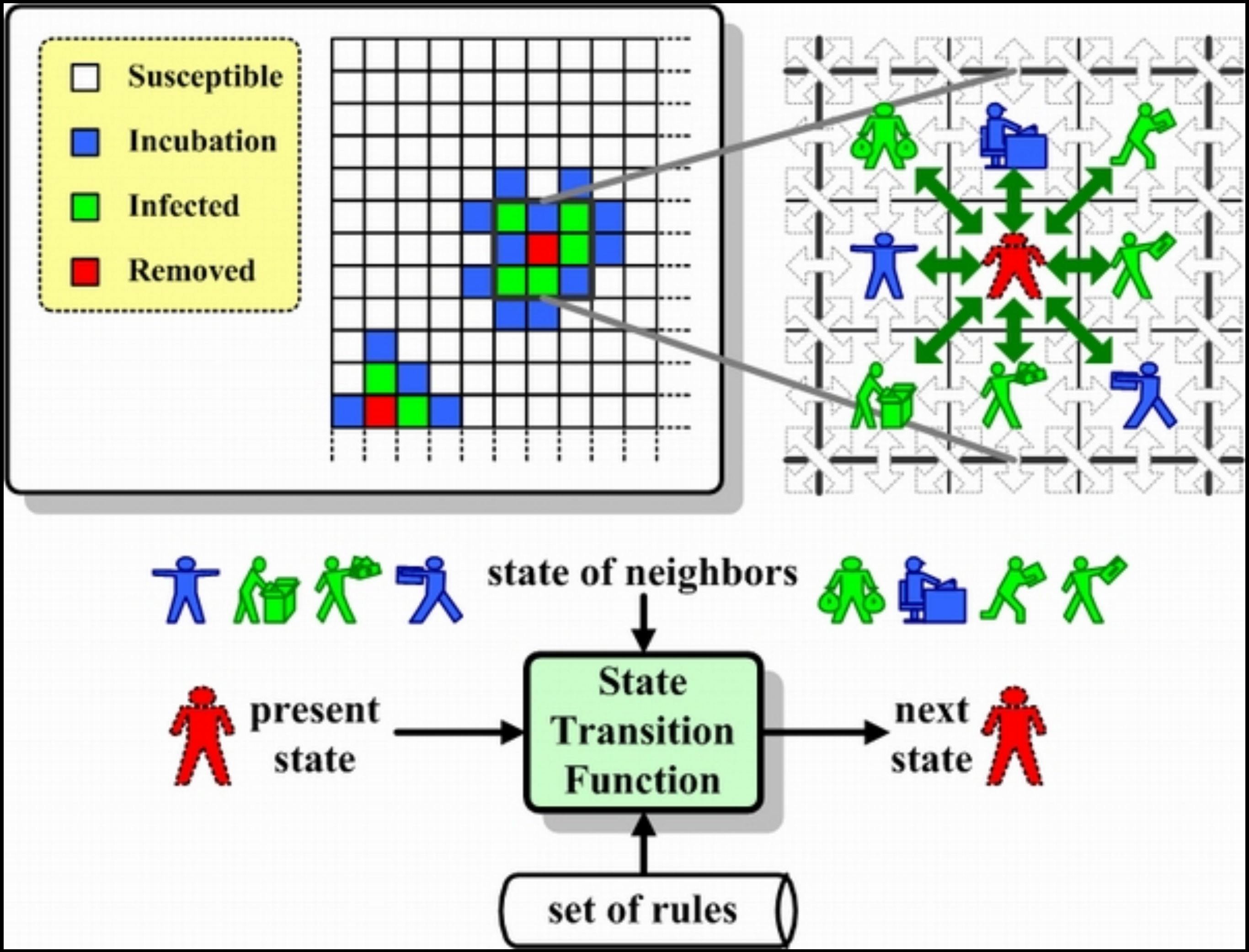
CAs as Discrete Simulation Models

- Cellular Automata are discrete
  - Time changes in incremental steps
    - Differential equations for continuous time, typically
  - Space is represented explicitly, in regular arrangements of cells
  - Each cell is in one of a finite number of states at any given time
- Deterministic
  - Initial states of cells determine the rest of the computation
- Each of these assumptions can be relaxed

Cellular Automata

- Invented by John von Neumann (circa 1950)
- Discrete dynamical systems with interesting properties
  - Emergent behavior, Self-organized criticality, Percolation
  - Examples: Game of Life, forest fire model
- Discrete simulation models
  - Focus on complex systems where components interact in time AND space
  - Examples: Forest fires, fluid dynamics, ferro-magnetic modeling (ISING), epidemics, predator/prey interactions, enzyme reactions, gene expression, social interactions, economies, etc.

Example 1: Modeling Disease Spread Across a Landscape



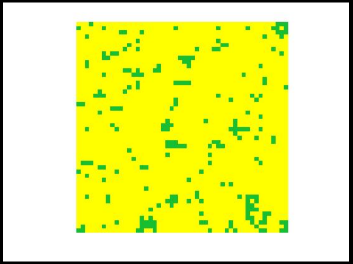
jasss.soc.surrey.ac.uk/7/4/2.html

Example 2: Forest Fire model

- Introduced by Henly (1988); Bak, Chen, and Tang (1990)
- Illustrates SOC, percolation
- Applications: Spread of forest fires, orchard blight, infectious diseases, oil movement through porous rock, and many more

Fire CA

- A grid of cells in one of 3 states:
  - Yellow is *empty*
  - Green is *tree*
  - Red is *burning*



- Each cell changes state depending on the states of its neighbors (including itself)
- All cells are updated simultaneously
  - *How can this be?*
- Define initial conditions and iterate the model synchronously

Suppose we want to develop a CA model of forest fire spread

1. Problem statement
2. Define model states and variables
3. Establish transition rules and neighborhood
4. Deal with grid boundaries (wraparound is easiest)
5. Initialize a grid (initial conditions)
6. Run the simulation with different parameters and observe behavior

1. Problem statement

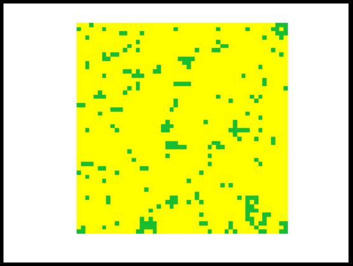
- Problem: How does fire spread across a landscape?
- Model landscape as a 2-D grid
  - Some grid cells have trees
  - Some trees are burning
  - Lightening strikes can cause new fires
  - [option: Some trees are resistant to burning]
- Goal: understand how different initial conditions, probabilities, landscape sizes, and time scales affect fire spread

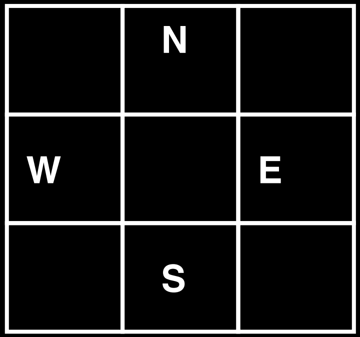
2. Identify States and Variables

- Establish correspondence between states in the problem statement and in the model:
  - 0 EMPTY
  - 1 TREE
  - 2 BURNING
- Establish variables to hold probabilities
  - Of a tree growing into an empty space p
  - Of an existing tree igniting spontaneously f
    - lightening, etc.

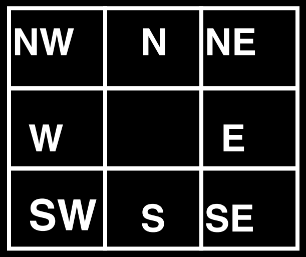
3. Define Transition Rules

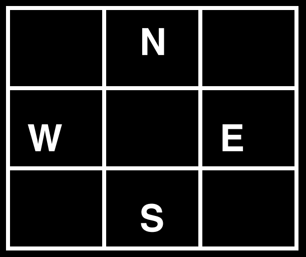
- 4 transition rules
  - A burning cell turns into an empty cell
  - A tree will burn if at least one neighbor is burning
  - A tree ignites with probability *f* even if no neighbor is burning
  - An empty space fills with a tree with probability *p*
- Neighborhood (von Neumann)





Moore vs. Von Neumann Neighborhood





Moore

Von Neumann

4. Define grid boundaries

MATLAB Hint: For wraparound use mod operator and define arrays Self, North, South, East, West

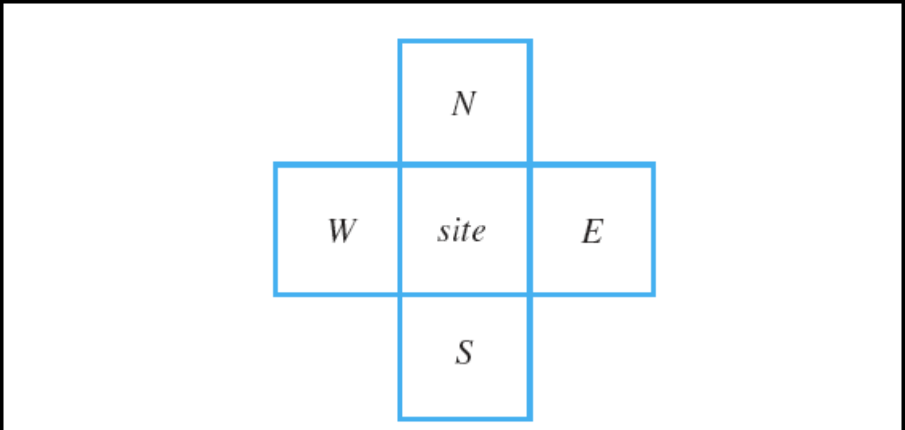
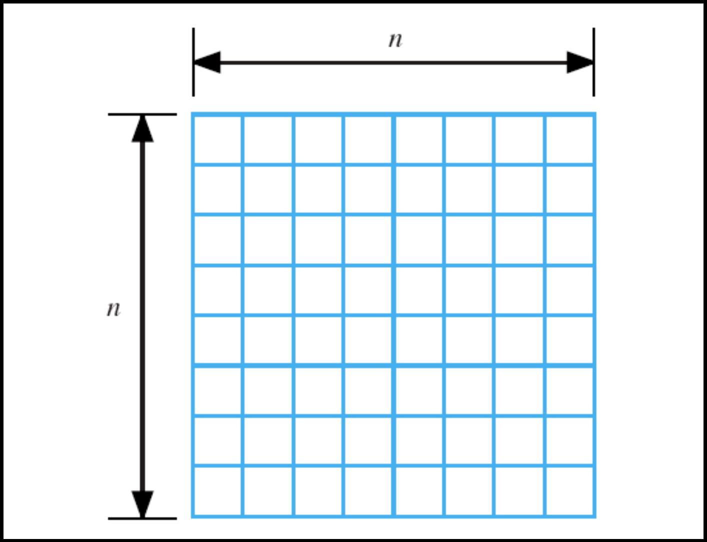


Figure 11.2.2 Cells that determine a site's next value



5. Initialize the Grid

- Define one color for each state
- Determine size of grid
- Determine initial state of grid
  - For FF model, we assume random initialization

MATLAB Hints

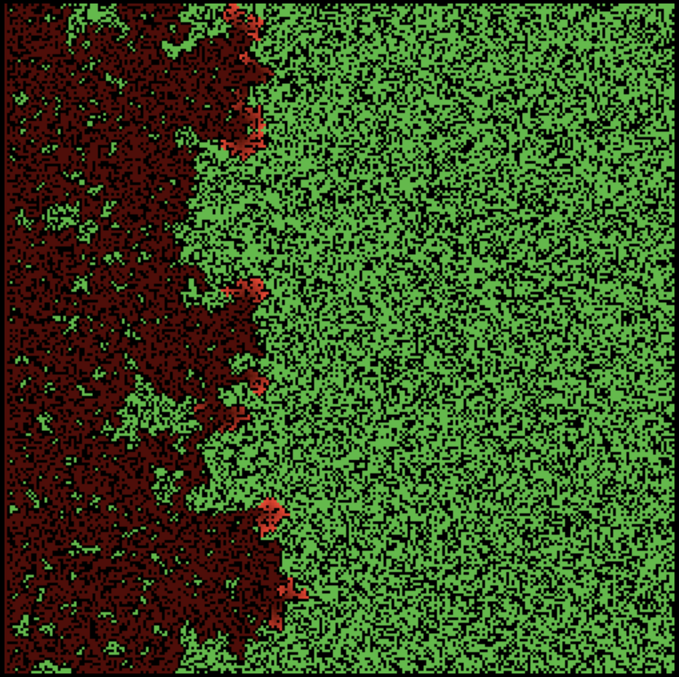
Use colormap, rand, floor, image, and axis functions to intialize the grid

```
% help colormap for other built-in maps
colormap(gray(5))

% returns a 3x3 matrix of random numbers between 0 and 1
rand(3)

%randgrid is  a 3x3 matrix with values between 1 and 5
%floor rounds down to the nearest integer
randgrid = floor(rand(3)*5 + 1)

%image uses the 5 colors in the colormap to display the grid
image(randgrid)
axis off                    %removes axis labels
axis square                 % makes a square grid
```

Warmup Exercise

- Watch Video 1
- Implement a CA that grows a spiral

6. Run the Model



http://ccl.northwestern.edu/netlogo/models/models/Sample%20Models/Earth%20Science/Fire.png

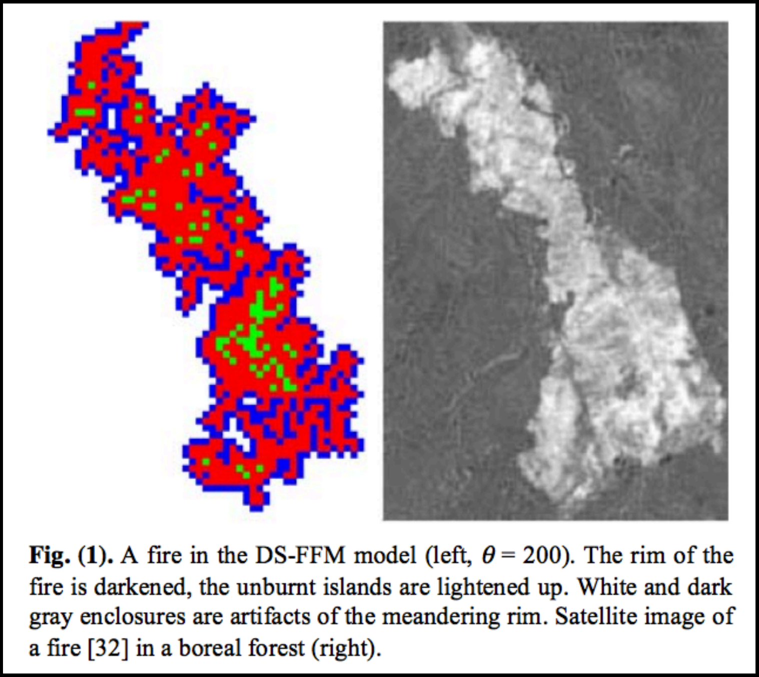
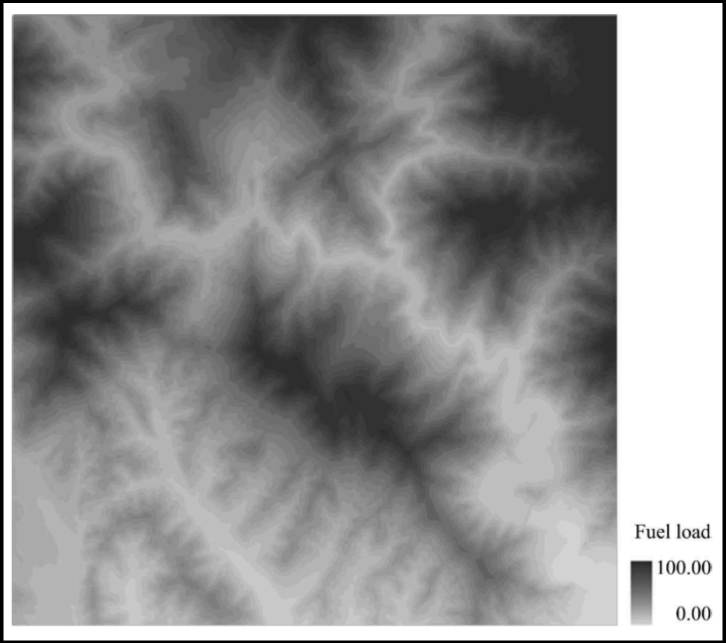


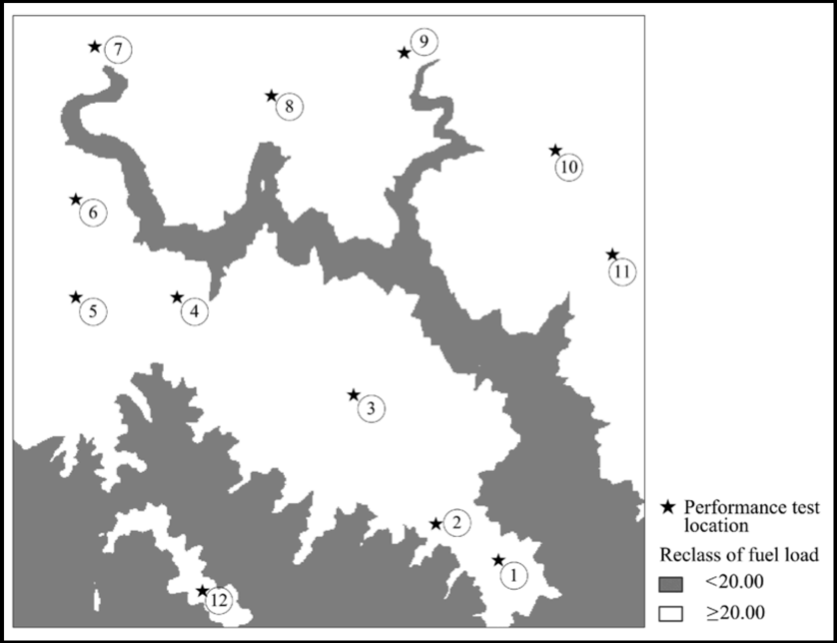
Fig. (1). A fire in the DS-FFM model (left, $\theta = 200$). The rim of the fire is darkened, the unburnt islands are lightened up. White and dark gray enclosures are artifacts of the meandering rim. Satellite image of a fire [32] in a boreal forest (right).

R.D. Zinck1,2 and V. Grimm "**More Realistic than Anticipated: A Classical Forest-Fire Model from Statistical Physics Captures Real Fire Shapes "*he Open Ecology Journal,* 2008, *1,* 8-13**

A framework of integrating GIS and parallel computing for spatial control problems – a case study of wildfire control

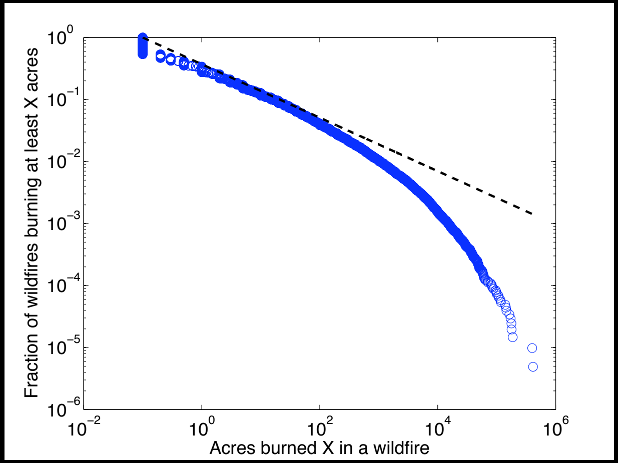
Yin et al. International Journal of Geographical Information Science (2012)

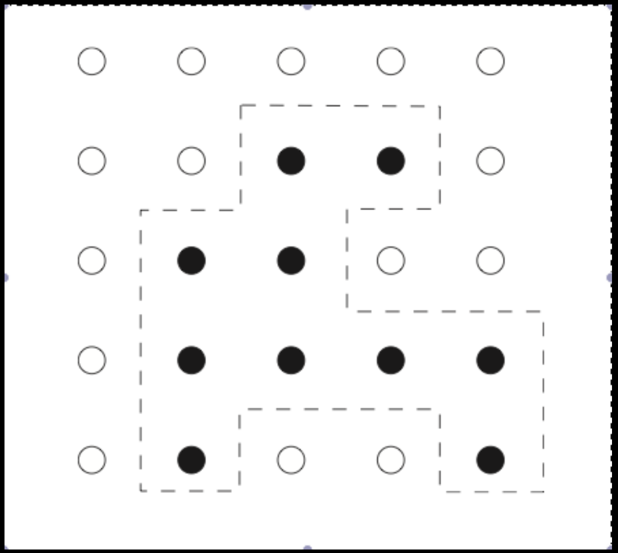




Cas as Discrete Dynamical Systems

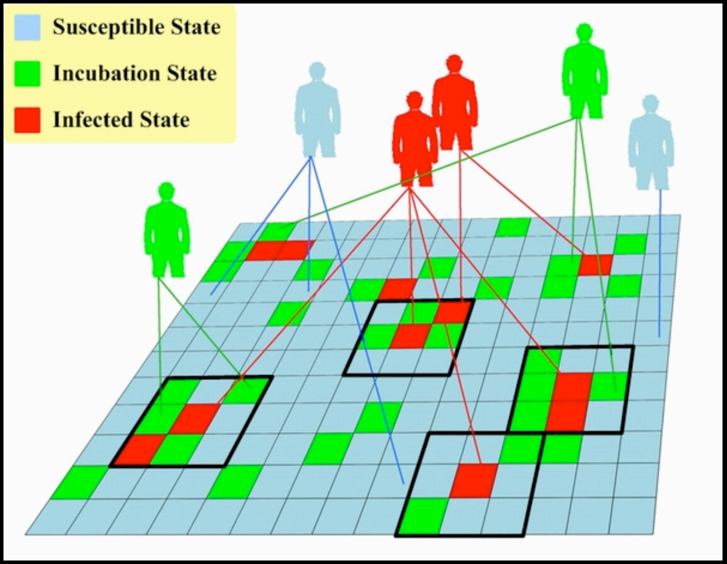
- Vary parameters and initial conditions
- Self-organized criticality
  - p/f controls behavior
  - Cluster sizes power-law distributed
  - What is a cluster size?
- Percolation
  - Submerge a porous rock in a bucket of water. Does the center of the rock get wet?
  - Release a rat in a maze. What is the probability that it finds its way out, if it searches randomly?

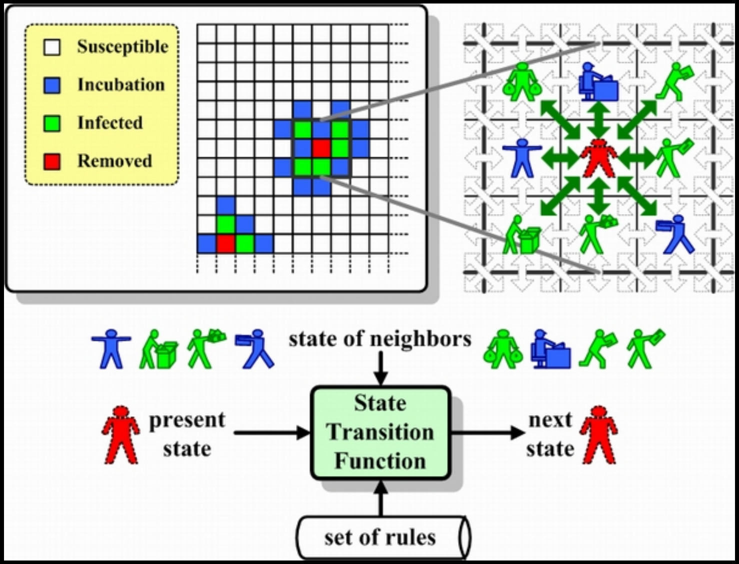




http://www.nt.ntnu.no/users/skoge/prost/proceedings/acc11/data/papers/1503.pdf

Return to Disease Example





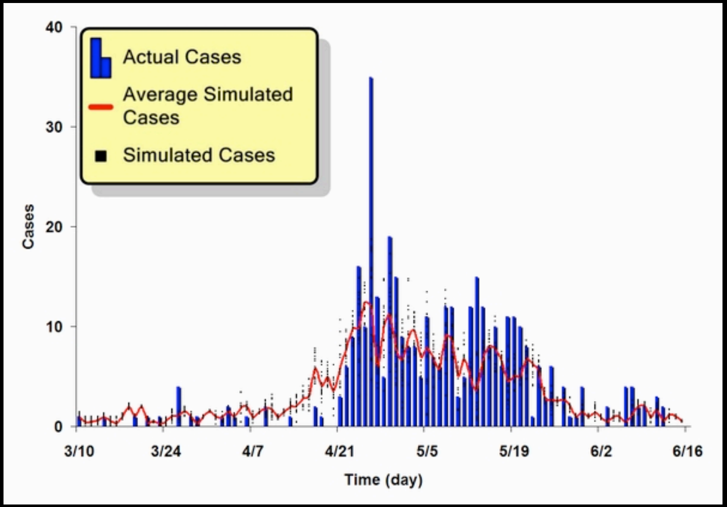


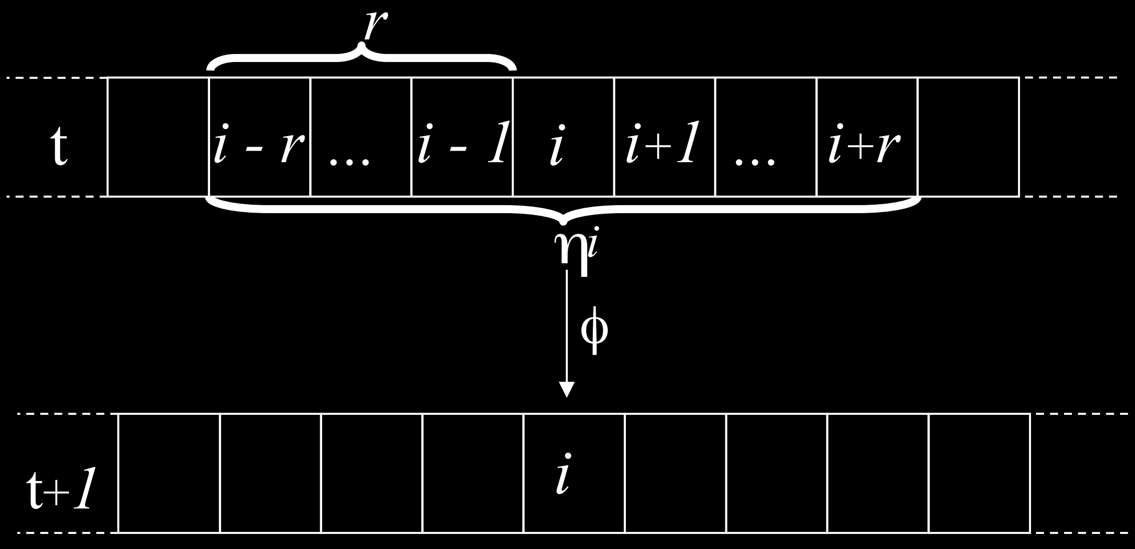
Figure 8. A comparison of actual and simulated epidemic results for the SARS outbreak in Taipei

jasss.soc.surrey.ac.uk/7/4/2.html

A More Careful Description of CA *Preliminaries*

- State space: A set of possible values that represent the key attributes (state) of a system at a particular point in time
- In CS, a configuration of discrete states is used to model machines, i.e., <N,A,S,G> where
  - N is a finite set of states, e.g., {0,1}
  - A is a set of transitions between states
    - e.g., {0 => 1, 1 =>0}
  - S is a start set (subset of N)
  - G is a stop state (subset of N)
- In physics, a continuous model of a physical system:
  - States are vectors (e.g., position, momentum)
  - Transitions are expressed as matrices or differential (difference) equations
- Phase space
  - Representation of all possible states of system
  - Phase diagram: All possible trajectories through phase space in time

One-Dimensional CA



- Linear array of identical cells (called lattice), each of which can be in a finite number of k states
- The local state of cell i at time t is denoted:

$$s_t^i \in \Sigma = \{0,1,\ldots,k-1\}$$

- The global state s$_t$ at time *t* is the *configuration* of the entire array,

$$\mathrm{s}_t = (s_t^0, s_t^1, \ldots, s_t^{N-1}) \in \Sigma^N$$

- Where N is the size of the array

One-dimensional CA cont.

- At each time step, all cells in the array update their state simultaneously, according to a *local update rule* : $\phi : \eta \rightarrow s$
- This update rule takes as input the *local neighborhood configuration* η of a cell
- η consists of s_i and its 2r nearest neighbors (r cells on either side): $\eta_i = (s^{i-r}, \ldots, s, \ldots, s^{i+r})$
- *r* is called the *radius* of the CA
- The local update rule φ, which is the same for every cell in the array, can be represented as a lookup table, which lists all possible neighborhood configurations $s_{t+1}^{i} = \phi(\eta_t^i)$

Example One-Dimensional CA Rule 110

- The number of states, *k*=2.
- The alphabet Σ = {0,1}
- The size of the array, N=11.
- The configuration space

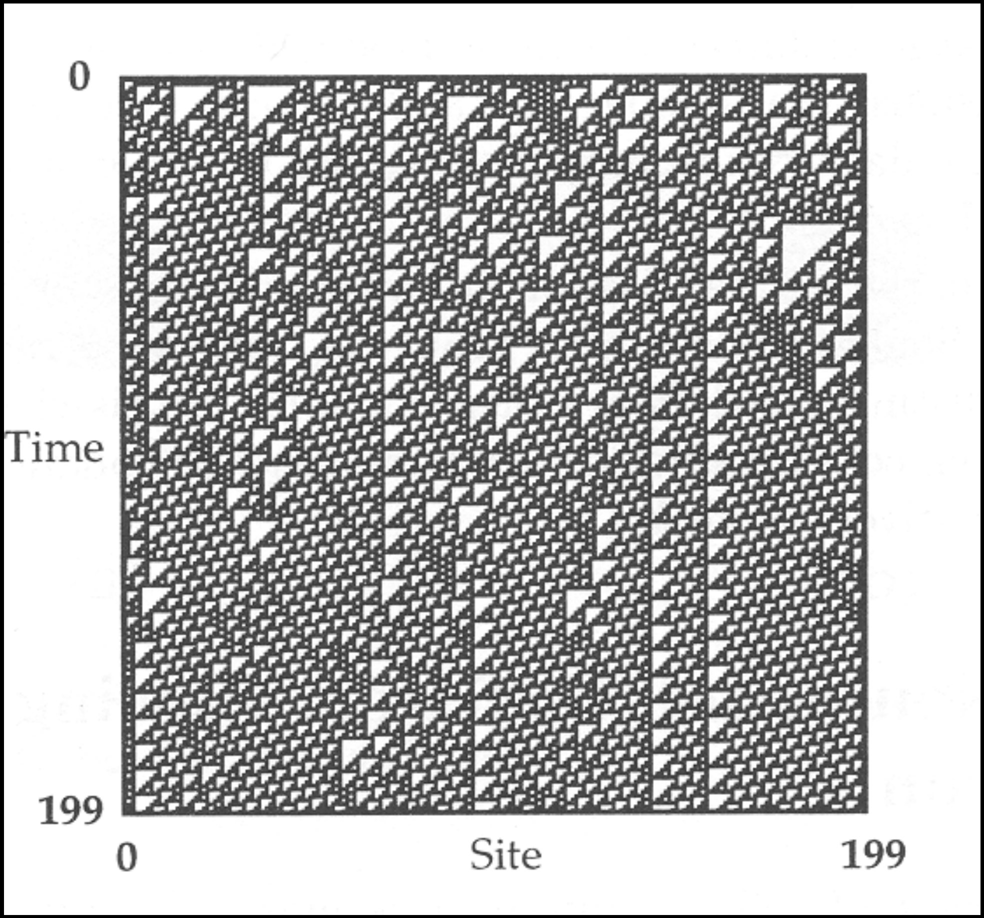
$$\Sigma^N = \{(0,0,0,0,0,0,0,0,0,0,0),(0,0,0,0,0,0,0,0,0,0,11),...\}$$

- The radius *r* = 1.
- The rule table : $s^i_{t+1} = \phi(\eta^i_t)$

– neighborhood	000	001	010	011	100	101	110	111
– ϕ	0	1	1	1	0	1	1	0

- This is rule 110 (base 10) because the output states are: 01101110 (base 2). Read right to left. Known as Wolfram notation.
- Rule 110 supports universal computation.

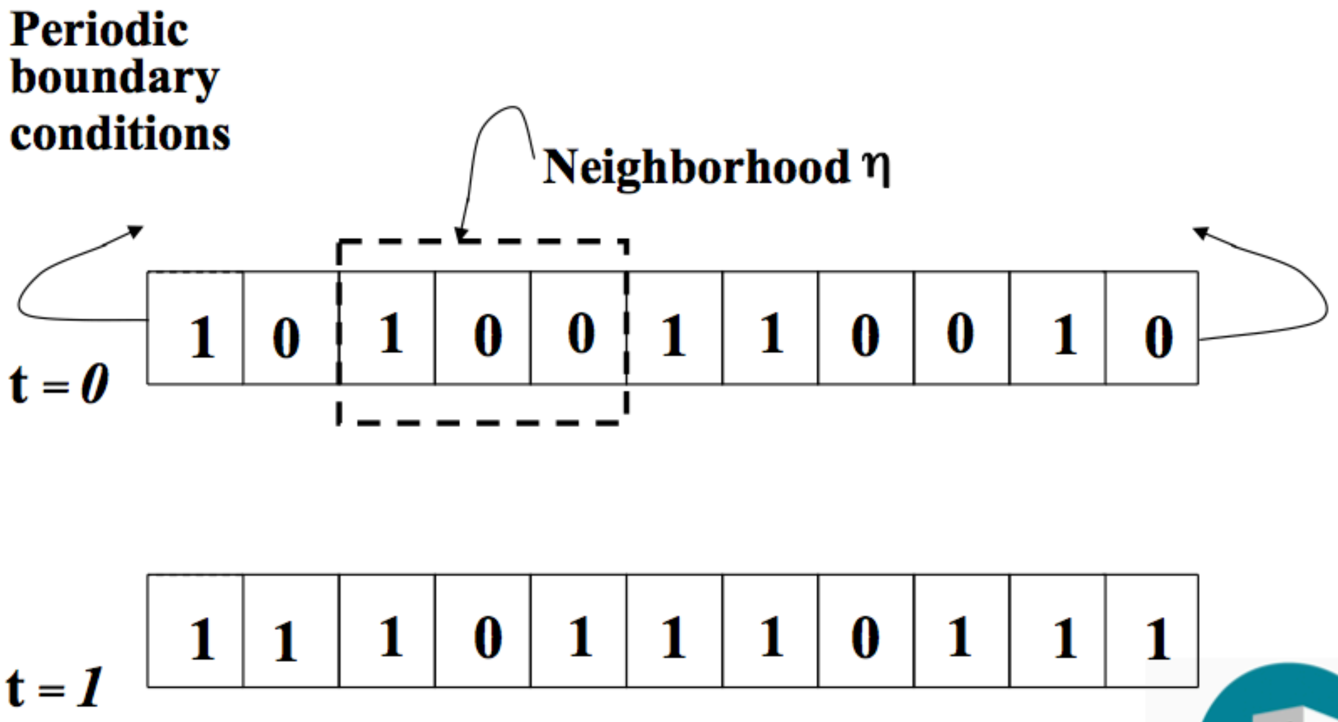
Rule 110 space-time plot



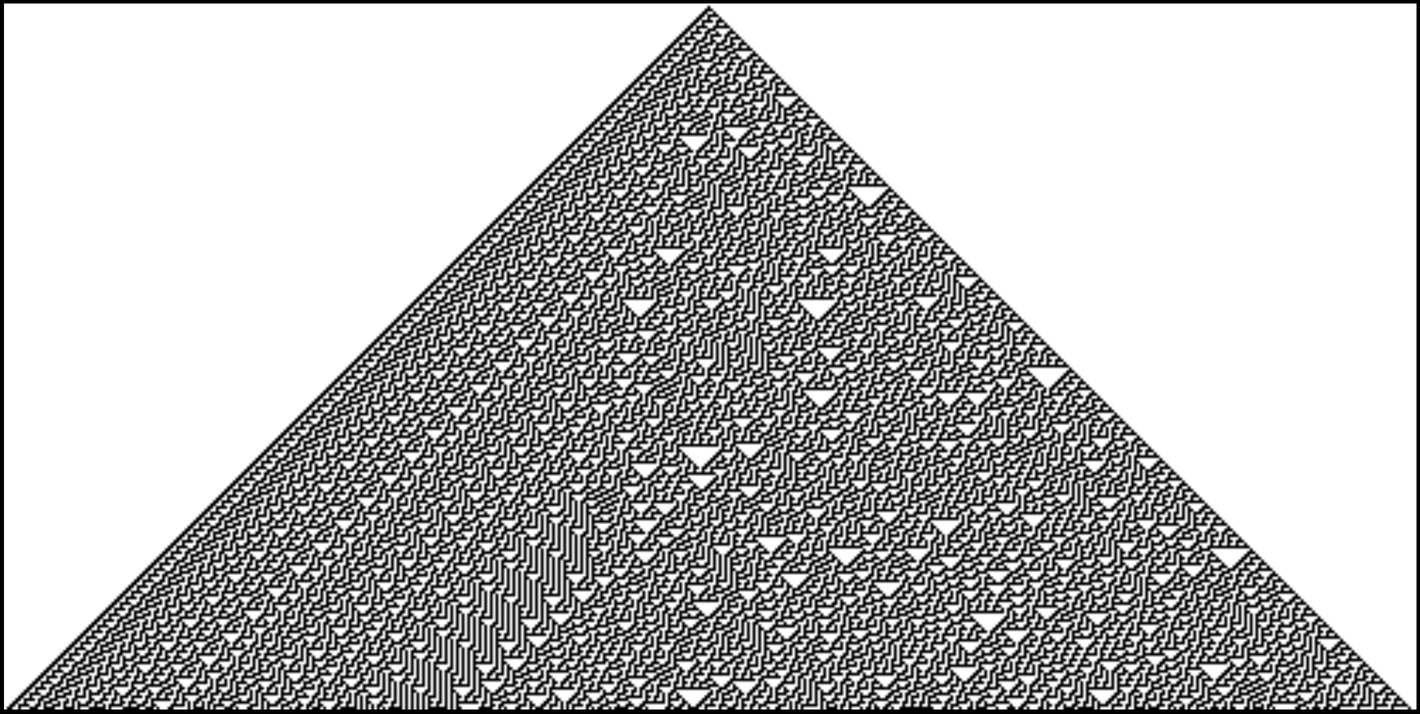
Rule Table ϕ :

Neighborhood:	000	001	010	011	100	101	110	111
Output bit:	0	1	1	1	0	1	1	0

Lattice:



Rule 30



current pattern	111	110	101	100	011	010	001	000
new state for center cell	0	0	0	1	1	1	1	0

Comments on Rule 30

- Generates apparent randomness, despite being finite
- Wolfram proposed using the central column as a psuedo-random number generator (png)
- Passes many tests for randomness, but many inputs produce regular patterns:
  - All zeroes
  - 00001000111000 repeated infinitely
- Used in Mathematica for creating random integers (Wikipedia)

What does a 1-D CA Model?

- View nature as an information processing system
- CAs as an idealized model of nature
  - Digital state
  - Uniform physics
  - Local computation and communication
  - Exhibits complex behavior not obvious from direct inspection of the rules
  - Easily visualized
- Supports universal computation, which some think is common in nature
  - Is there an upper limit on the complexity of computation in nature?
  - Recent work by Valiant on what functions are learnable by evolution

Computation in Cellular Automata

- CAs as computers
  - Initial configuration constitutes the data that the physical computer is processing
  - Transition function implements the algorithm which is applied to the data
  - Examples: Majority calculations, synchronization (Mitchell and Crutchfield)
- CAs as logical universes within which computers can be embedded
  - Initial configuration constitutes a computer
  - Transition function is the "physics" obeyed by the parts of the embedded computer
  - The algorithm and data are functions of the initial configuration of the embedded computer
  - In the most general case, the initial configuration is a universal computer
  - Examples: Game of life as a universal computer, Fredkin's billiard ball computer

CAs can exhibit complex behavior

Wolfram's Classification

- Class I: Eventually every cell in the array settles into one state, never to change again
  - Analogous to computer programs that halt after a few steps and to dynamical systems that have fixed-point attractors

Wolfram Class I

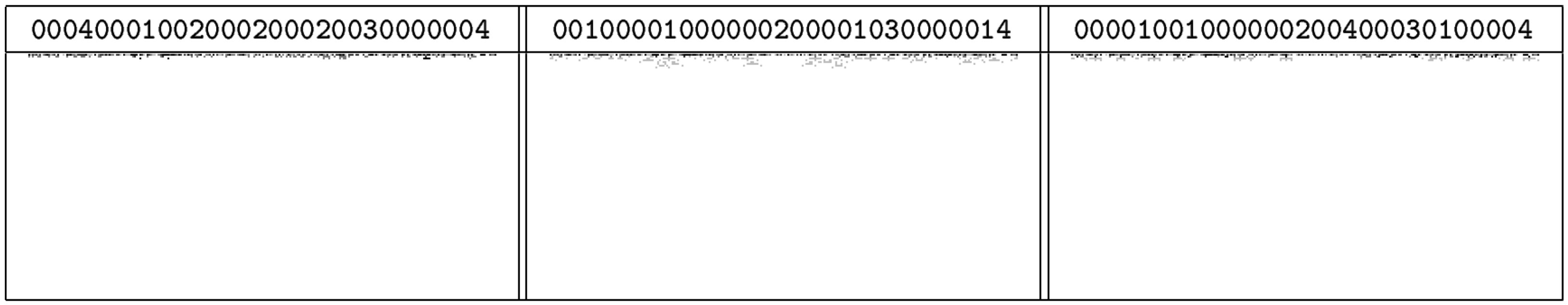


Figure 15.5 Examples of Wolfram's Class I

CA can exhibit complex behavior Wolfram's Classification

- Class I: Eventually every cell in the array settles into one state, never to change again
  - Analogous to computer programs that halt after a few steps and to dynamical systems that have fixed-point attractors
- Class II: Eventually the array settles into a periodic cycle of states, called a limit cycle
  - Analogous to computer programs that execute infinite loops and to dynamical systems that fall into limit cycles

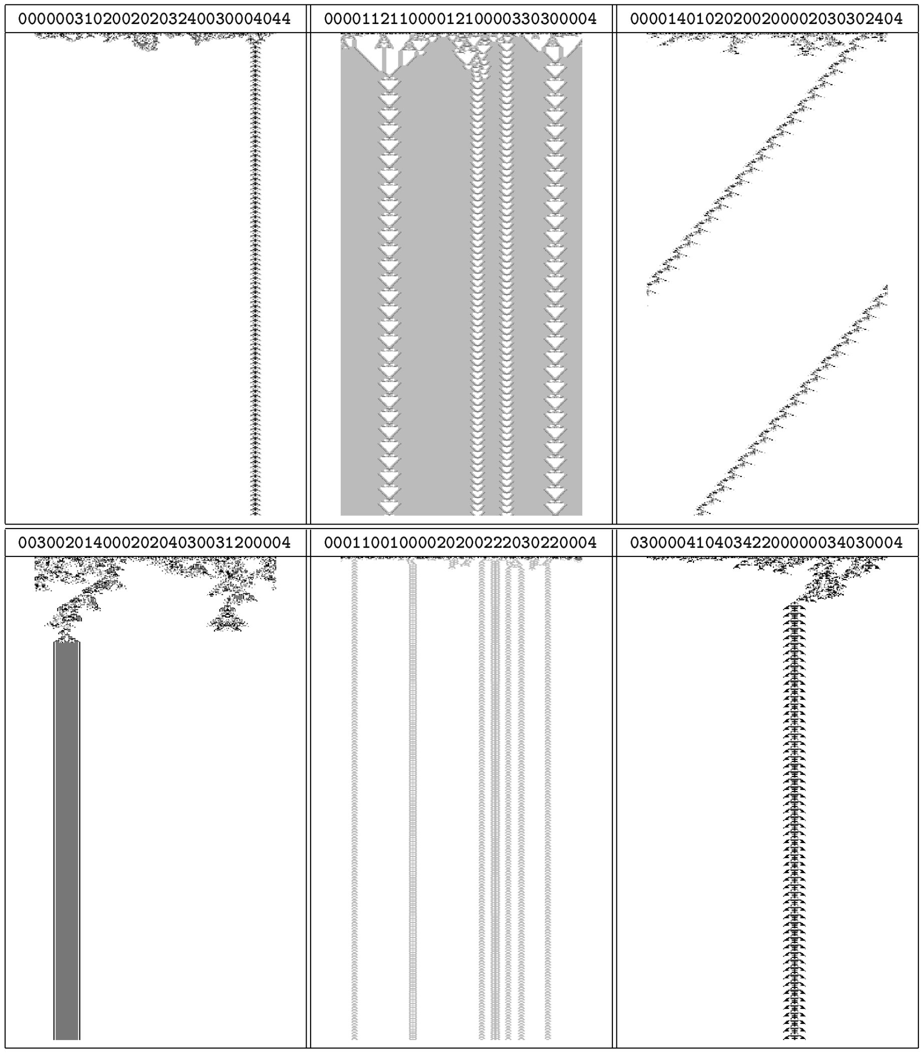


Figure 15.6 Examples of Wolfram's Class II

The Universi

CA can exhibit complex behavior Wolfram’s Classification

- Class I: Eventually every cell in the array settles into one state, never to change again
  - Analogous to computer programs that halt after a few steps and to dynamical systems that have fixed-point attractors
- Class II: Eventually the array settles into a periodic cycle of states, called a limit cycle
  - Analogous to computer programs that execute infinite loops and to dynamical systems that fall into limit cycles
- Class III: The array forms “aperiodic” random-like patterns
  - Analogous to computer programs that are pseudo-random number generators (pass most tests for randomness, highly sensitive to seed, or initial condition).
  - Analogous to chaotic dynamical systems. Almost never repeat themselves, sensitive to initial conditions, embedded unstable limit cycles

Wolfram’s Class III

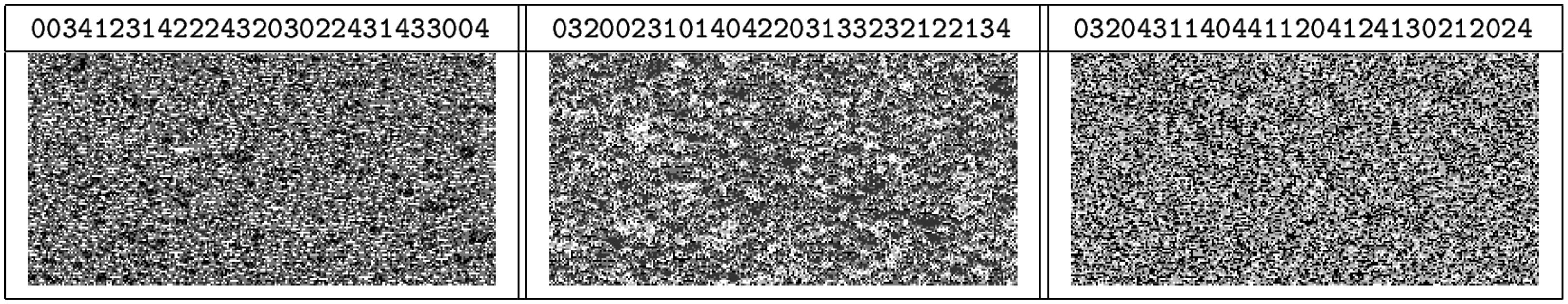


Figure 15.7 Examples of Wolfram’s Class III

CA can exhibit complex behavior Wolfram's Classification

- Class I: Eventually every cell in the array settles into one state, never to change again
  - Analogous to computer programs that halt after a few steps and to dynamical systems that have fixed-point attractors
- Class II: Eventually the array settles into a periodic cycle of states, called a limit cycle
  - Analogous to computer programs that execute infinite loops and to dynamical systems that fall into limit cycles
- Class III: The array forms "aperiodic" random-like patterns
  - Analogous to computer programs that are pseudo-random number generators (pass most tests for randomness, highly sensitive to seed, or initial condition).
  - Analogous to chaotic dynamical systems. Almost never repeat themselves, sensitive to initial conditions, embedded unstable limit cycles
- Class IV: The array forms *complex* patterns with localized structure that move through space and time
  - Difficult to describe. Not regular, not periodic, not random
  - Speculate: this is *interesting* computation, the edge of chaos
  - Example: Rule 110

Wolfram’s Class IV

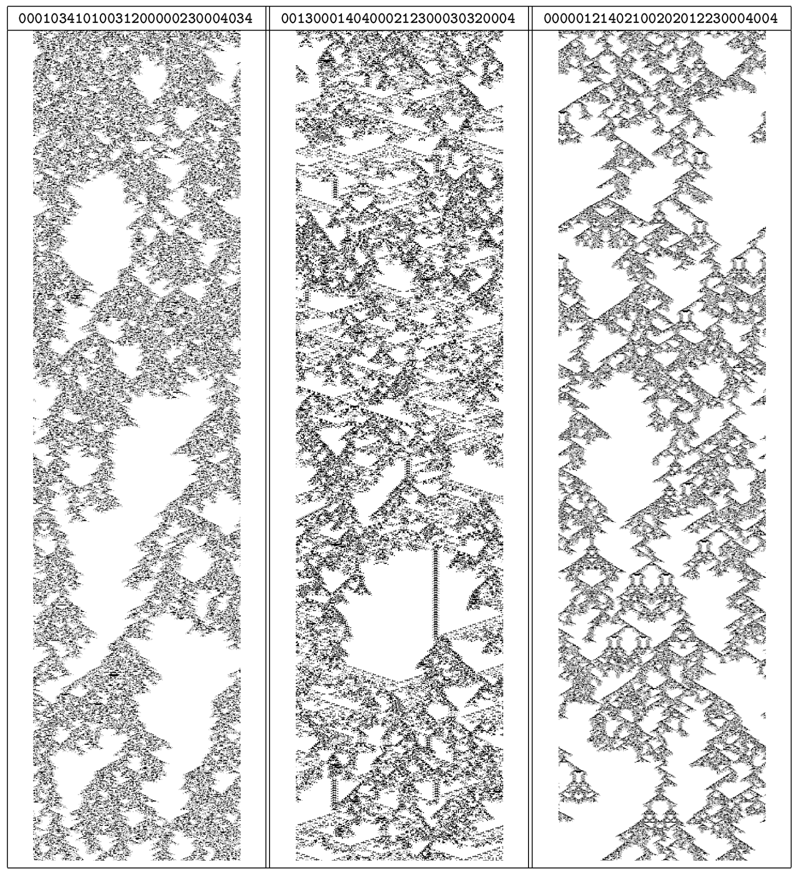


Figure 15.8 Examples of Wolfram’s Class IV

Can you predict which Cas will have which class of behavior?

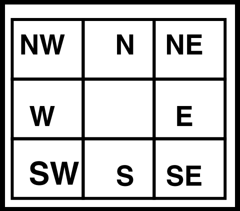
Comment on Wolfram's Rule 110

"However, there is nonetheless a distinct limit to the complexity produced by class 4 automata. The many images of such automata in Wolfram's book all have a similar look to them, and although they are nonrepeating, they are interesting (and intelligent) only to a degree. Moreover, they do not continue to evolve into anything more complex, nor do they develop new types of features. Once could run these automata for trillions of trillions of iterations and the image would remain at the same limited level of complexity. They do not EVOLVE into, say, insects or humans or Chopin preludes or anything else that we might consider of a higher order of complexity than the streaks and intermingling triangles displayed in these images."

Kurtzwil *A Theory of Technological Evolution*, pp. 85-95

The Game of Life
John Conway (1970)

- The number of states, k = 2
- The alphabet, Σ = {0,1}   // {‘dead’,’alive’}
- The Moore neighborhood



- Transition rules:
  - Loneliness: A live cell with less than 2 live neighbors, dies
  - Overcrowding: A live cell with more than 3 live neighbors, dies
  - Birth: A dead cell with exactly 3 live neighbors becomes a live cell
  - Survival: A live cell with 2 or 3 live neighbors stays alive

Example Transition

0	1	0
0	0	1
0	1	0

$\Rightarrow$ 1

- Center square changes to one (birth)
- “just 3 for birth, 2 or 3 for survival”

Possible Life Histories
(Dynamical Behaviors)

- Static structures
  - Behive, loaf, pond,etc. (Fig. 15.11)
- Periodic structures
  - Blinkers (Fig. 15.12)
- Moving structures
  - Gliders
- Glider guns
- Logical gates (and, or, not)
- Self-reproducing structures

Static Objects

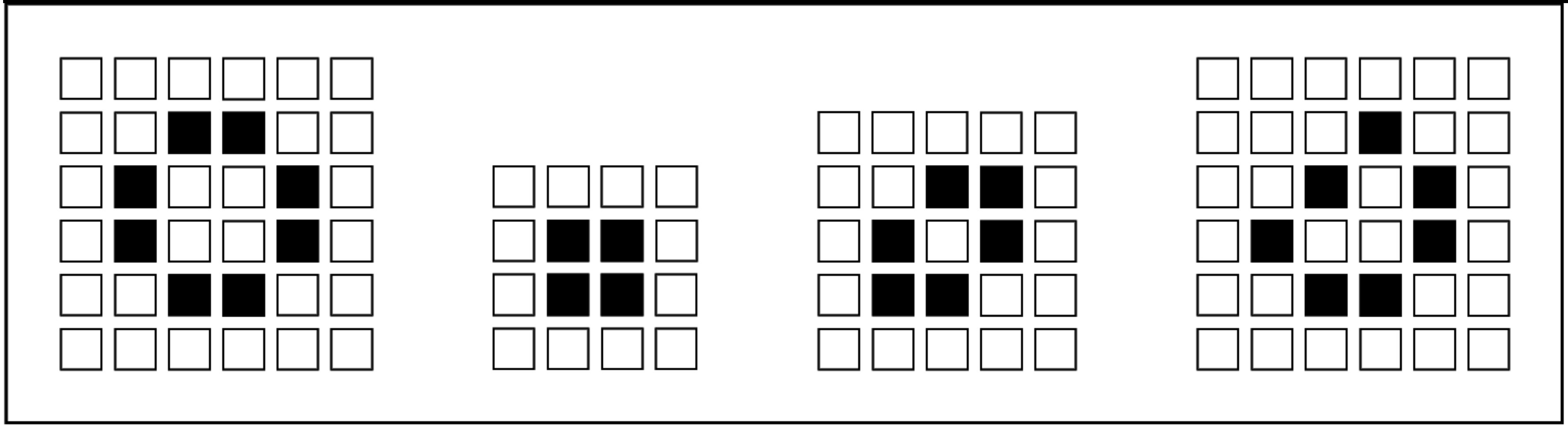
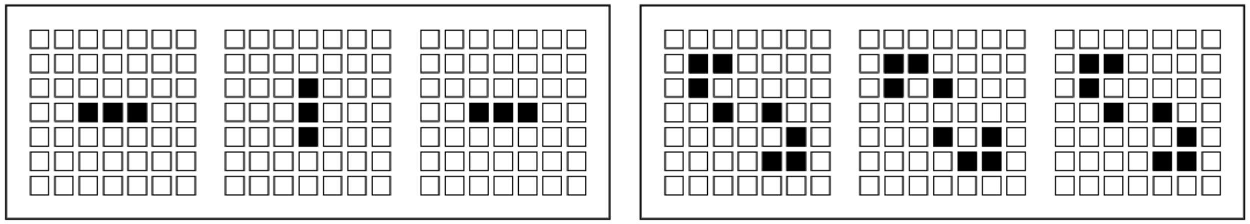


Figure 15.11 Examples of static objects in Conway's Game of Life

Periodic Objects



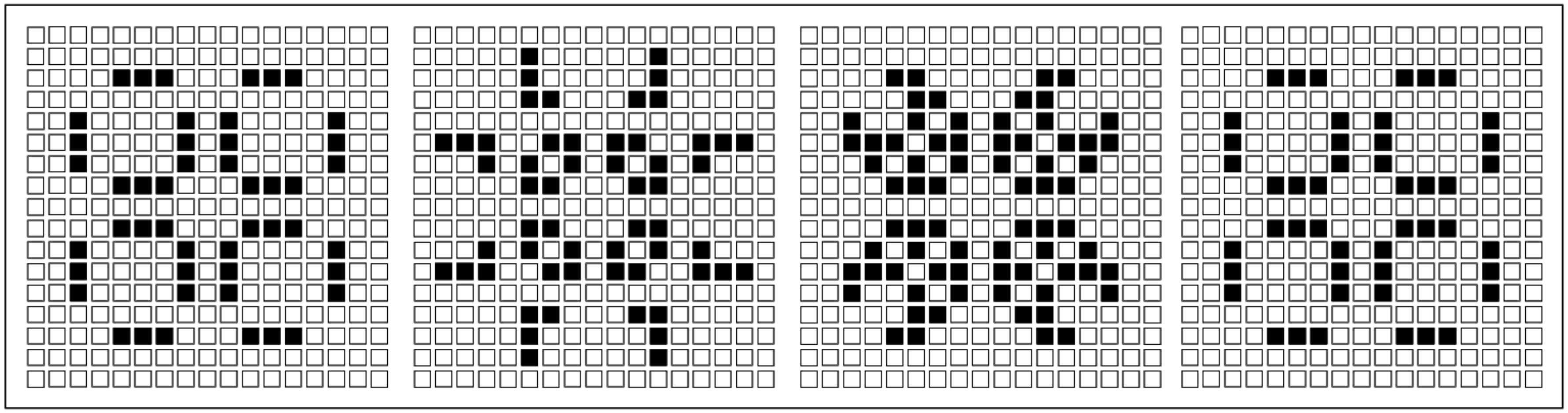
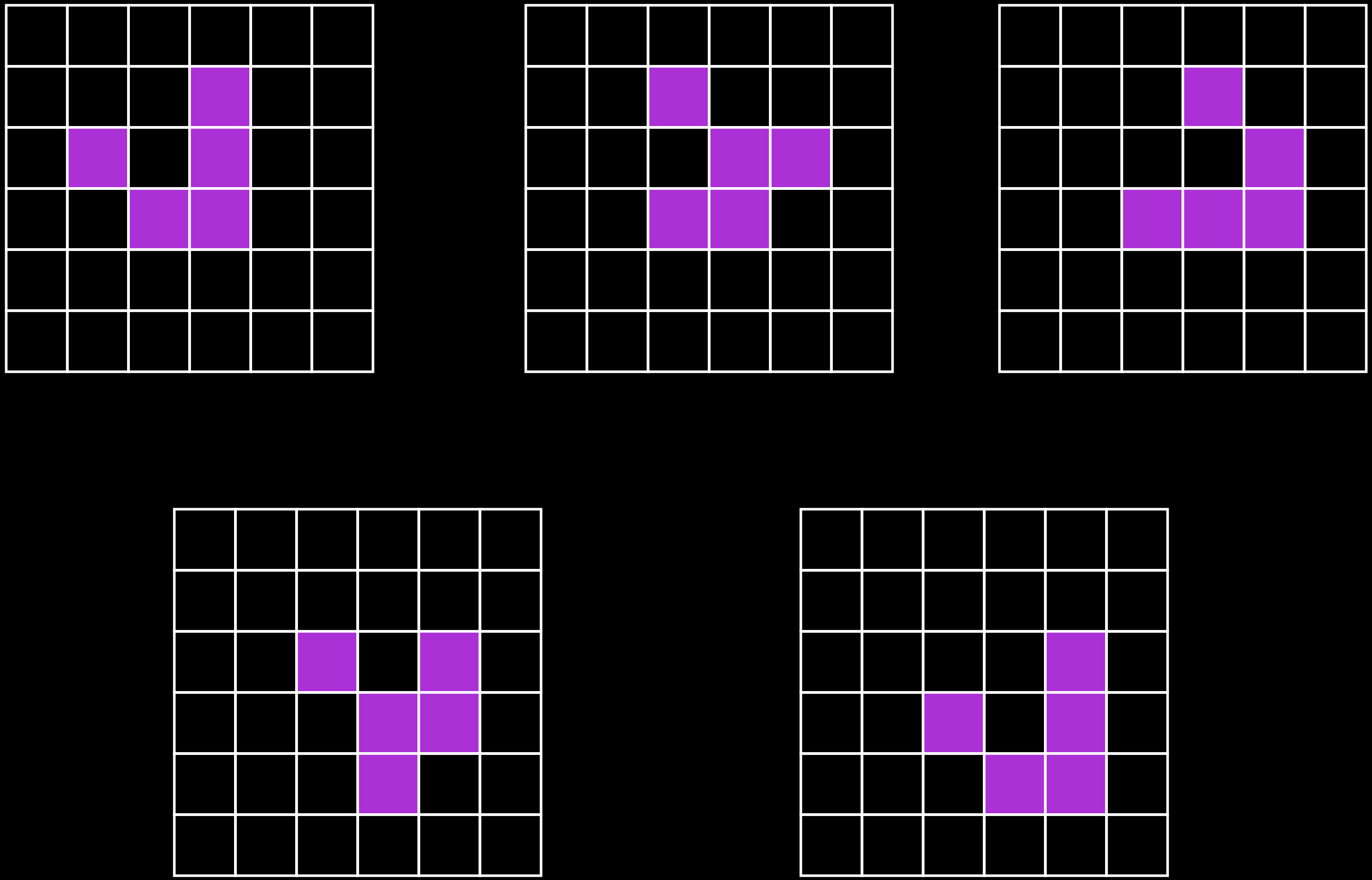


Figure 15.12 Examples of simple periodic objects in Conway's Game of Life

Gliders (moving objects)



Can we predict the dynamics of a given initial condition?

- Consider a straight line of n live cells

– n = 1,2	Dies out immediately
– n = 3	Blinker
– n = 4	Becomes a beehive
– n = 5	Traffic lights
– n = 6	Dies out at t= 12
– n = 7	Interesting behavior, terminating in the honey farm
– n = 8	4 blocks and 4 beehives
– n = 9	2 sets of traffic lights
– etc.	

Langton's Lambda Parameter

- Wolfram's classification scheme is *phenomenological*
  - Argument by visual inspection of space-time diagrams
- Chris Langton (1986) quantified the classification scheme by introducing the parameter λ
- λ is a statistic of the output states in the CA lookup table, defined as the fraction of non-quiescent states in this table
- The quiescent state is an arbitrarily chosen state, e.g., for a 2-state CA, we might choose $s = 0$ as the quiescent state

Lambda Parameter cont.

$$\lambda = \frac{|\eta| - n_q}{|\eta|}, 0 \le \lambda \le 1.0 - \frac{1}{k}$$

Lambda Parameter cont.

$$\lambda = \frac{|\eta| - n_q}{|\eta|}, 0 \le \lambda \le 1.0 - \frac{1}{k}$$

- N_q = the number of rules that map to the quiescent state
- Using the lambda parameter to study CAs
  - Table walkthrough: start with one random table and progressively perturb it through the range
  - Random table: Interpret lambda as a bias on the random selectino of states to fill up the table (get a new table for every new value of lambda).
- Use statstics to measure CA “average” behavior, as a function of lambda
  - Single-site entropy
  - Two-site mutual information
  - Same site mutual information across time steps

Lambda Parameter cont.

- Table walkthrough illustrates interesting change in dynamics as a function of lambda
  - Transient lengths increase dramatically in the middle of the range (next slide)
  - However, different traversals of lambda space using walkthrough method make the transition at different lambda values, although there is a well defined distribution around a mean value.

Lambda Space and Wolfram Classes

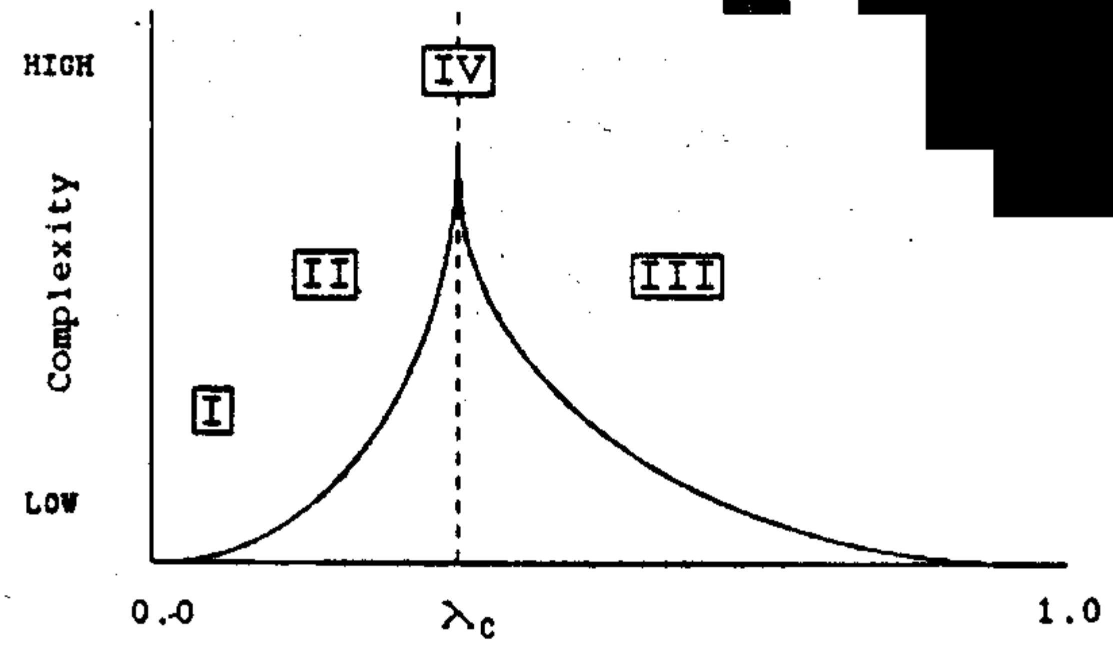


Fig. 16. Location of the Wolfram classes in λ space.

Critical Slowing Down

Length of transients depends on array size

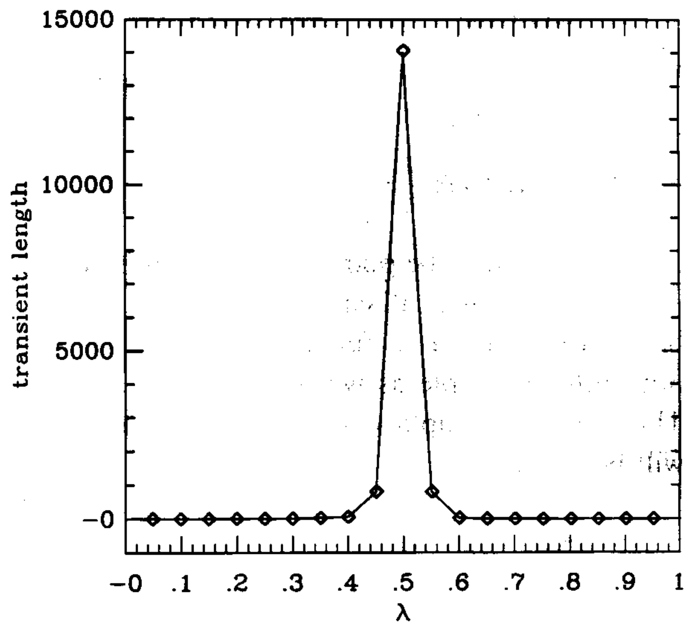


Fig. 3. Average transient length as a function of λ in an array of 128 cells.

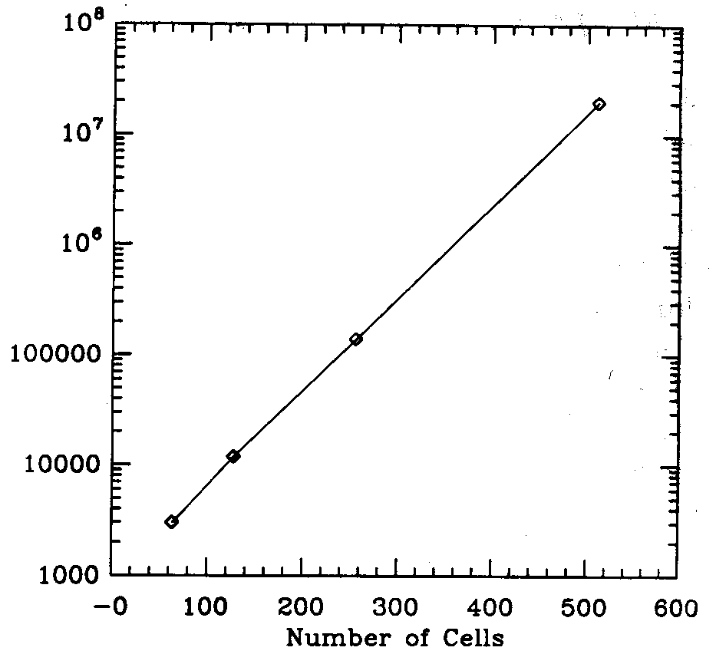


Fig. 4. Growth of average transients as a function of array size for $\lambda = 0.50$.

Comments on Lambda Parameter

- Claims
  - There is a phase transition between periodic and chaotic behavior. Most complex behavior is in the vicinity of the transition: the “edge of chaos.”
  - CAs near the transition point correspond to Wolfram’s Class IV
  - CAs capable of performing complex computations will be found near the transition point (long transients)
  - E.g., the Game of life has $\lambda = 0.273$
    - in the transition region for k=2, N=9 2-D CAs
- Criticism of the lambda parameter:
  - CAs with high lambda value can still have simple behavior. Lambda describes “average” behavior
  - Lambda does not take the initial state of the computation into account