

# Chapter 15

## Complexity

We introduce cellular automata models, neural networks, genetic algorithms, and explore the concepts of self-organization and complexity.

### 15.1 Cellular Automata

Part of the fascination of physics is that it allows us in many cases to reduce natural phenomena to a few simple laws. It is perhaps even more fascinating to think about how a few simple laws can produce the enormously rich behavior that we see in nature. In this chapter we will discuss several models that illustrate some of the new ideas that are emerging from the study of “complex systems.”

The first class of models we discuss are known as *cellular automata*. Cellular automata were originally introduced by von Neumann and Ulam in 1948 as an idealization of biological self-reproduction, and are examples of discrete dynamical systems that can be simulated exactly on a digital computer. A cellular automaton can be thought of as a checkerboard with colored squares (the cells). Each cell changes its color at the tick of an external clock according to a rule based on the present configuration (microstate) of the cells in its neighborhood.

More formally, cellular automata are mathematical idealizations of dynamical systems in which space and time are discrete and the quantities of interest have a finite set of discrete values that are updated according to a local rule. The important characteristics of cellular automata include the following:

1. Space is discrete, and there is a regular array of sites (cells). Each site has a finite set of values.
2. Time is discrete, and the value of each site is updated in a sequence of discrete time steps.
3. The rule for the new value of a site depends only on the values of a *local* neighborhood of sites near it.

t:	111	110	101	100	011	010	001	000
t + 1:	0	1	0	1	1	0	1	0

Figure 15.1: Example of a local rule for the time evolution of a one-dimensional cellular automaton. The variable at each site can have values 0 or 1. The top row shows the  $2^3 = 8$  possible combinations of three sites. The bottom row gives the value of the central site at the next time step. This rule is termed 01011010 in binary notation (see the second row), the modulo-two rule, or rule 90. Note that 90 is the base ten (decimal) equivalent of the binary number 01011010, that is,  $90 = 2^1 + 2^3 + 2^4 + 2^6$ .

4. The variables at each site are updated *simultaneously* (“synchronously”) based on the values of the variables at the previous time step.

Because the original motivation for studying cellular automata was their biological aspects, the lattice sites frequently are referred to as cells. More recently, cellular automata have been applied to a wide variety of physical systems ranging from fluids to galaxies. We will refer to sites rather than cells, except when we are explicitly discussing biological systems.

We first consider one-dimensional cellular automata with the neighborhood of a given site assumed to be the site itself and the sites immediately to the left and right of it. Each site also is assumed to have two states (a Boolean automata). An example of such a rule is illustrated in Fig. 15.1, where we see that a rule can be labeled by the binary representation of the update for each of the eight possible neighborhoods and by the base ten equivalent of the binary representation. Because any eight digit binary number specifies an one-dimensional cellular automata, there are  $2^8 = 256$  possible rules.

Program `ca1` takes as input the decimal representation of the rule and produces the rule matrix (array `update`). This array is used to update each site on the lattice using periodic boundary conditions. On a single processor computer, it is necessary to use an additional array so that the state of each site can be updated using the previous values of the sites in its local neighborhood. The state of the sites as a function of time is shown on the screen with time running downwards.

```
PROGRAM ca1
! one-dimensional Boolean cellular automata
DIM update(0 to 7),site(0 to 501)
CALL setrule(update())
CALL initial(site(),L,tmax,#2)
CALL iterate(site(),L,update(),tmax,#2)
END

SUB setrule(update())
  INPUT prompt "rule number = ": rule
  OPEN #1: screen 0,0.5,0.2,0.8
  SET BACKGROUND COLOR "black"
  SET COLOR "white"
  FOR i = 7 to 0 step -1
```

```

    LET update(i) = int(rule/2^i) ! find binary representation
    LET rule = rule - update(i)*2^i
    LET bit2 = int(i/4)
    LET bit1 = int((i - 4*bit2)/2)
    LET bit0 = i - 4*bit2 - 2*bit1
    ! show possible neighborhoods
    PRINT using "#": bit2,bit1,bit0;
    PRINT " ";
NEXT i
PRINT
FOR i = 7 to 0 step -1
    PRINT using "##": update(i); ! print rules
    PRINT " ";
NEXT i
CLOSE #1
END SUB

SUB initial(site(),L,tmax,#2)
RANDOMIZE
OPEN #2: screen 0.5,1,0.1,0.9
ASK PIXELS px,py
SET WINDOW 1,px,py,1
SET COLOR "yellow"
LET L = 2*int(px/8) - 8
LET tmax = L
LET site(L/2) = 1 ! center site
BOX AREA 1+2*L,2*L+4,1,4 ! each site 4 x 4 pixels
END SUB

SUB iterate(site(),L,update(),tmax,#2)
! update lattice
! need to introduce additional array, sitenew, to temporarily
! store values of newly updated sites
DIM sitenew(0 to 501)
FOR t = 1 to tmax
    FOR i = 1 to L
        LET index = 4*site(i-1) + 2*site(i) + site(i+1)
        LET sitenew(i) = update(index)
        IF sitenew(i) = 1 then BOX AREA 1+i*4,i*4+4,1+t*4,t*4+4
    NEXT i
    MAT site = sitenew
    LET site(0) = site(L) ! periodic boundary conditions
    LET site(L+1) = site(1)
NEXT t
END SUB

```

The properties of all 256 one-dimensional cellular automata have been cataloged (see Wolfram). We explore some of the properties of one-dimensional cellular automata in Problems 15.1 and 15.2.

*Problem 15.1.* One-dimensional cellular automata

- a. Use Program `ca1` and rule 90 shown in Fig. 15.1. This rule also is known as the “modulo-two” rule, because the value of a site at step  $t + 1$  is the sum modulo 2 of its two neighbors at step  $t$ . Choose the initial configuration to be a single nonzero site (seed) at the midpoint of the lattice. It is sufficient to consider the time evolution for approximately twenty steps. Is the resulting pattern of nonzero sites self-similar? If so, characterize the pattern by a fractal dimension.
- b. Consider the properties of a rule for which the value of a site at step  $t + 1$  is the sum modulo 2 of the values of its neighbors *plus* its own value at step  $t$ . This rule is termed rule 10010110 or rule  $150 = 2^1 + 2^2 + 2^4 + 2^7$ . Start with a single seed site.
- c. Choose a random initial configuration for which the independent probability for each site to have the value 1 is 50%; otherwise, the value of a site is 0. Consider the time evolution of rule 90, rule 150, rule  $18 = 2^1 + 2^4$  (00010010), rule  $73 = 2^0 + 2^3 + 2^6$  (01001001), and rule 136 (10001000). How sensitive are the patterns that are formed to changes in the initial conditions? Does the nature of the patterns depend on the use or nonuse of periodic boundary conditions?

Because the dynamical behavior of many of the 256 one-dimensional Boolean cellular automata is uninteresting, we also consider one-dimensional Boolean cellular automata with larger neighborhoods. The larger neighborhood implies that there are many more possible update rules, and it is convenient to place some reasonable restrictions on the rules. First, we assume that the rules are symmetric, for example, the neighborhood 100 produces the same value for the central site as 001. We also assume that the zero neighborhood 000 yields 0 for the central site, and that the value of the central site depends only on the sum of the values of the sites in the neighborhood, for example, 011 produces the same value for the central site as 101 (Wolfram 1984).

A simple way of coding the rules that is consistent with these requirements is as follows. Call the size of the neighborhood  $z$  if the neighborhood includes  $2z + 1$  sites. Each rule is labeled by  $\sum_m a_m 2^m$ , where  $a_m = 1$  if the central cell is 1 when the sum of all values in the neighborhood equals  $m$ ; else  $a_m = 0$ . As an example, take  $z = 2$  and suppose that the central site equals one when two or four sites are unity. This rule is labeled by  $2^2 + 2^4 = 20$ .

*Problem 15.2.* More one-dimensional cellular automata

- a. Modify Program `ca1` so that it incorporates the possible rules discussed in the text for a neighborhood of  $2z + 1$  sites. How many possible rules are there for  $z = 1$ ? Choose  $z = 1$  and a random initial configuration, and determine if the long time behavior for each rule belongs to one of the following classes:
  - (a) A homogeneous state where every site is either 0 or 1. An example is rule 8.
  - (b) A pattern consisting of separate stable or periodic regions. An example is rule 4.
  - (c) A chaotic, aperiodic pattern. An example is rule 10.
  - (d) A set of complex, localized structures that may not live forever. There are no examples for  $z = 1$ .

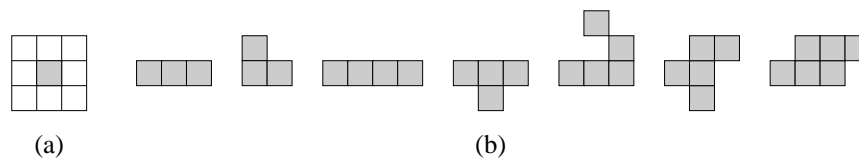


Figure 15.2: (a) The local neighborhood of a site is given by the sum of its eight neighbors. (b) Examples of initial configurations for the Game of Life, some of which lead to interesting patterns. Live cells are shaded.

- b. Modify your program so that  $z = 2$ . Wolfram (1984) claims that rules 20 and 52 are the only examples of complex behavior (class 4). Describe how the behavior of these two rules differs from the behavior of the other rules. Determine the fraction of the rules belonging to the four classes.
- c. Repeat part (b) for  $z = 3$ .
- d. Assume that sites can have three values, 0, 1, and 2. Classify the behavior of the possible rules for the case  $z = 1$ .

The results of Problem 15.2 suggest that an important feature of cellular automata is their capability for “self-organization.” In particular, the class of complex localized structures is distinct from regular as well as aperiodic structures. This intermediate structure is the focus of *complexity theory* whose goal is to explain complex phenomena in nature.

One-dimensional models are too limited to study the complexity of nature, and we now consider several two-dimensional models. The philosophy is the same except that the neighborhood contains more sites. Program `ca2` sets up the rule matrix and updates sites using the eight neighbor sites shown in Fig. 15.2a. There are now  $2^9 = 512$  possible configurations for the eight neighbors and the center site, and  $2^{512}$  possible rules. Clearly, we cannot go through all these rules in any systematic fashion as we did for one-dimensional cellular automata. For this reason, we will set up our rule matrix based on other considerations.

The rule matrix incorporated in Program `ca2` implements the best known two-dimensional cellular automata model: the *Game of Life*. This model, invented in 1970 by the mathematician John Conway, produces many fascinating patterns. The rules of the game are simple. For each cell determine the sum of the values of its four nearest and four next-nearest neighbors (see Fig. 15.2a). A “live” cell (value 1) remains alive only if this sum equals 2 or 3. If the sum is greater than 3, the cell will “die” (become 0) at the next time step due to overcrowding. If the sum is less than 2, the cell will die due to isolation. A dead cell will come to life only if the sum equals 3.

```
PROGRAM ca2
LIBRARY "csgraphics"
DIM update(0 to 511),cell(50,50)
CALL setrule(update(),L)
LET flag$ = ""
DO
```

```

CALL initial(cell(),L)
DO
  CALL iterate(cell(),update(),L)
  IF key input then
    GET KEY k
    IF (k = ord("s")) or (k = ord("S")) then
      LET flag$ = "stop"
    END IF
  END IF
  LOOP UNTIL k <> 0
  LET k = 0
LOOP until flag$ = "stop"
END

SUB setrule(update(),L)
! rule for Game of Life
FOR i = 0 to 511
  LET update(i) = 0
NEXT i
! three neighbors alive
FOR nn1 = 0 to 5
  FOR nn2 = nn1+1 to 6
    FOR nn3 = nn2+1 to 7
      LET index = 2^nn1 + 2^nn2 + 2^nn3
      LET update(index) = 1 ! center dead
      LET update(index+256) = 1 ! center alive
    NEXT nn3
  NEXT nn2
NEXT nn1
! two neighbors and center alive
FOR nn1 = 0 to 6
  FOR nn2 = nn1+1 to 7
    LET index = 256 + 2^nn1 + 2^nn2
    LET update(index) = 1
  NEXT nn2
NEXT nn1
SET BACKGROUND COLOR "black"
SET COLOR "white"
INPUT prompt "lattice size = ": L
CALL compute_aspect_ratio(L,xwin,ywin)
SET WINDOW -0.2*xwin,1.2*xwin,-0.2*ywin,1.2*ywin
END SUB

SUB initial(cell(),L)
FOR i = 1 to L
  FOR j = 1 to L

```

```

        LET cell(i,j) = 0
        SET COLOR "yellow"
        BOX AREA i,i+1,j,j+1
        SET COLOR "black"
        BOX LINES i,i+1,j,j+1
    NEXT j
NEXT i
SET CURSOR 1,1
! click on cell to change its state or outside of lattice
! to update cells
SET COLOR "white"
PRINT "click on cell to toggle or outside of lattice to continue."
DO
    GET POINT x,y
    IF x > 1 and x < L and y > 1 and y < L then
        LET i = truncate(x,0)
        LET j = truncate(y,0)
        IF cell(i,j) = 0 then
            SET COLOR "black"
            BOX AREA i,i+1,j,j+1
            LET cell(i,j) = 1
        ELSE
            SET COLOR "yellow"
            BOX AREA i,i+1,j,j+1
            LET cell(i,j) = 0
            SET COLOR "black"
            BOX LINES i,i+1,j,j+1
        END IF
    END IF
LOOP until x < 1 or x > L or y < 1 or y > L
SET CURSOR 1,1
SET COLOR "white"
PRINT "Hit any key for new lattice, 's' to stop";
PRINT "                                "
END SUB

SUB iterate(cell(,),update(),L)
    DIM cellnew(50,50)
    FOR i = 1 to L
        FOR j = 1 to L
            CALL neighborhood(cell(,),i,j,sum,L)
            LET cellnew(i,j) = update(sum)
            IF cell(i,j) = 1 and cellnew(i,j) = 0 then
                SET COLOR "yellow"
                BOX AREA i,i+1,j,j+1
                SET COLOR "black"
            END IF
        NEXT j
    NEXT i
END SUB

```

```

        BOX LINES i,i+1,j,j+1
    ELSE IF cell(i,j) = 0 and cellnew(i,j) = 1 then
        SET COLOR "black"
        BOX AREA i,i+1,j,j+1
    END IF
NEXT j
NEXT i
MAT cell = cellnew
END SUB

SUB neighborhood(cell(,),i,j,sum,L)
    LET ip = i + 1
    LET im = i - 1
    LET jp = j + 1
    LET jm = j - 1
    IF i = 1 then
        LET im = L
    ELSE IF i = L then
        LET ip = 1
    END IF
    IF j = 1 then
        LET jm = L
    ELSE IF j = L then
        LET jp = 1
    END IF
    LET sum = cell(i,jp) + 2*cell(i,jm) + 4*cell(im,j)
    LET sum = sum + 8*cell(ip,j) + 16*cell(ip,jp) + 32*cell(ip,jm)
    LET sum = sum + 64*cell(im,jp) + 128*cell(im,jm) + 256*cell(i,j)
END SUB

```

Program `ca2` allows the user to use any update rule by changing `SUB setrule`. Program `ca2` has not been optimized for the Game of Life and is written so that it can be easily modified for any cellular automata rule.

*Problem 15.3.* The Game of Life

- Program `ca2` allows the user to determine the initial configuration interactively. Choose several initial configurations with a small number of live cells and investigate the different types of patterns that emerge. Some suggested initial configurations are shown in Fig. 15.2b. Does it matter whether you use fixed or periodic boundary conditions?
- Modify Program `ca2` so that each cell is initially alive with a 50% probability. What types of patterns typically result after a long time? What happens for 20% live cells? What happens for 70% live cells?
- Assume that each cell is initially alive with probability  $p$ . Given that the density of live cells at time  $t$  is  $\rho(t)$ , what is  $\rho(t+1)$ , the expected density at time  $t+1$ ? Do the simulation and plot  $\rho(t+1)$  versus  $\rho(t)$ . If  $p = 0.5$ , what is the steady-state density of live cells?

- d. As we found in part (b), the system will develop structure even if each cell is randomly populated at  $t = 0$ . One measure of the increasing order in the system has been introduced by Schulman and Seiden and is analogous to the entropy in statistical mechanics. The idea is to divide the  $L \times L$  system into boxes of linear dimension  $b$  and determine  $n_i$ , the number of live cells in the  $i$ th box. The quantity  $S$  is given by

$$S = \frac{1}{L^2} \log_2 \prod_{i=1}^{(L/b)^2} \binom{b^2}{n_i}. \quad (15.1)$$

The argument of the logarithm in (15.1) is the total number of microscopic states associated with a given sequence of  $n_i$ . Roughly speaking,  $S$  measures the extent to which live cells are correlated. Determine  $S$  as a function of time starting from a 50% random configuration. First consider  $L = 50$  and  $b = 2, 3, 4$ , and 5. Average over at least ten separate runs. Describe how the behavior of the entropy depends on the level of “coarse graining” determined by the value of  $b$ . Does  $S$  decrease monotonically with time? Does it reach an equilibrium value? Increase  $L$  and the number of independent runs, and repeat your averages.

The Game of Life is an example of a universal computing machine. That is, we can set up an initial configuration of live cells to represent any possible program and any set of input data, run the Game of Life, and in some region of the lattice the output data will appear. The proof of this result (see Berlekamp et al.) involves showing how various configurations of cells represent the components of a computer including wires, storage, and the fundamental components of a CPU — the digital logic gates that perform **and**, **or**, and other logical and arithmetic operations.

*Problem 15.4.* Other two-dimensional cellular automata

- a. Consider a Boolean automaton with each site labeled by 1 (“on”) and 0 (“off”). We adopt an update rule such that the value of each site at time  $t + 1$  is determined by the vote of its four nearest neighbors (on a square lattice) at time  $t$ . The update rule is that a site becomes on if 2, 3, or 4 of its four neighbors are on. Consider initial configurations for which 1-sites occur with probability  $p$  and 0-sites occur with probability  $1 - p$ . Because the voting rule favors the growth of 1-sites, it is interesting to begin with a minority of 1-sites. Choose  $p = 0.1$  and determine what happens to isolated 1-sites. How do they grow initially? For what shape (convex or concave) does a cluster of 1-sites stop growing? What happens to clusters of 1-sites such as those shown in Fig. 15.3? (If necessary, create such a configuration.) Show that for  $p = 0.1$ , the system eventually freezes in a pattern made of 1-site rectangular islands in a sea of 0-sites. What happens for  $p = 0.14$ ? Can you define a “critical density”  $p_c$  at which the behavior of the system changes? Consider square lattices with linear dimension  $L = 128$  and  $L = 256$  (see Vichniac).
- b. Suppose that the update rule is determined by the sum of the center cell and its eight nearest and next-nearest neighbors. If this sum equals 4 or more, then the center site equals 1 at the next time step (see Vichniac). This rule also favors the growth of 1-sites and leads to a phenomenon similar to that found in part (a). Consider an initial configuration for which 1-sites occur with probability  $p$  and 0-sites occur with probability  $1 - p$ . Choose  $L = 128$  and show that for  $p = 0.2$ , the system eventually freezes. What is the shape of the 1-clusters? Show that if a single 0-site is changed to a 1-site at the surface of the largest 1-cluster, the cluster of

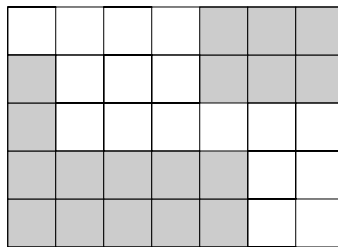


Figure 15.3: What is the evolution of these clusters using the rule discussed in Problem 15.4a. The shaded squares correspond to the 1-sites.

1-sites grows. What is the eventual state of the system? What is the behavior of the system for  $p = 0.3$ ? Is it possible to define a “critical density”  $p_c$  such that for  $p \geq p_c$ , the growth of the 1-clusters continues until all sites change to the 1-state? Consider larger values of  $L$  and show that the value of  $p_c$  appears to be insensitive to the value of  $L$ . What is your estimated value of  $p_c$  in the limit of an infinite lattice?

- c. There is one problem with the conclusions that you were able to reach in parts (a) and (b) — they are incorrect! The finite lattice results are misleading and  $p_c$  is zero in the limit of an infinite lattice. The probability of *any* configuration of 1-sites is unity for an infinite lattice. That is, somewhere in the lattice there is a “critical cluster” that will grow indefinitely until all sites in the lattice change to the 1-state. The moral of the story is, “Do not trust a simulation without a theory” (a paraphrase of a quote usually attributed to Eddington).

\**Problem 15.5.* Ising model cellular automata

- a. One of the most frequently studied models in statistical physics is the Ising model. In this model each cell has two states  $s_i$  that are labeled by  $\pm 1$  instead of 0 and 1. The energy is given by

$$E = -J \sum_{i,j=\text{nn}(i)}^N s_i s_j. \quad (15.2)$$

The notation  $j = \text{nn}(i)$  indicates that  $j$  is a nearest neighbor of  $i$ . If we think of  $s_i$  as representing the magnetic moment or spin at cell  $i$ , then the Ising model can be understood as a model of magnetism with  $J$  being the strength of the interaction between nearest neighbor spins. For  $J > 0$ , the lowest energy state occurs when all the spins are either all up ( $s_i = +1$ ) or all down ( $s_i = -1$ ). The magnetization  $m$  per spin is given by

$$m = \frac{1}{N} \sum_{i=1}^N s_i. \quad (15.3)$$

Monte Carlo algorithms for the simulation of the Ising model are discussed in Chapters ?? and ??.

In the cellular automata implementation of the Ising model, the energy of the entire lattice is fixed, and a spin may flip only if the energy of the lattice does not change. Such a situation occurs if a spin has two up neighbors and two down neighbors. Hence the update rule is to flip the spin at  $i$ , that is,  $s_i \rightarrow -s_i$ , if it has precisely two up neighbors; otherwise do not change  $s_i$ . But because we want to update all sites simultaneously, we have a problem. That is, if two neighboring spins have opposite signs and each has a total of two up neighbors, then flipping both spins would change the total energy. Why? The trick is to divide the lattice into two kinds of sites corresponding to the red and black squares of a checkerboard. First we update simultaneously all the red squares (hence keeping their neighbors, the black squares fixed), and then we update simultaneously all the black squares. Implement this cellular automata update of the Ising model by modifying `Program ca2` and write a subroutine to compute the mean magnetization per spin and the total energy per spin  $E/N$ . (The latter should not change with time.) The average is over the different configurations generated by the cellular automata updates. Use periodic boundary conditions.

- b. Compute the mean magnetization as a function of  $E/N$  for a square lattice with  $L = 20$ . One way to generate an initial configuration is to let each spin be up with probability  $p$ . Allow the system to “equilibrate” before accumulating values of the magnetization. Does the mean magnetization change from being close to zero to being close to unity as  $E/N$  is lowered? If such a qualitative change occurs, we say that there is a phase transition. To improve your results, average over many different initial configurations with the same value of  $E/N$  and determine the dependence of the mean magnetization on  $E/N$ . Because the same value of  $p$  will occasionally lead to different initial energies, write a subroutine that flips spins until the desired energy is obtained.
- c. Repeat part (b) for a  $40 \times 40$  lattice. Do your qualitative conclusions change?
- d. One difficulty with the cellular automata version of the Ising model is that it is not ergodic, that is, there are configurations with the same energy that cannot be reached for a given initial condition. In Chapter ?? we will see how to avoid this problem using Monte Carlo methods. Here we might allow the total energy  $E_0$  to vary a little, say  $\pm nJ$ , where  $n$  is an integer. During a run we can periodically flip a spin at random such that the total energy is in the range  $E_0 \pm nJ$ . Try this method for different values of  $n$ . Do your results for the mean magnetization change significantly?

`Program ca1` and `ca2` are inefficient due in part to the limitations imposed by the True BASIC language which does not have *bit* manipulation capability. The smallest element of computer memory contains a bit, which is a 0 or a 1. A *byte* is the size of memory needed to hold a single character, for example, a letter or a digit. More precisely, a byte is eight bits. Because there are  $2^8 = 256$  possible arrangements of 1’s and 0’s in a byte, a byte can represent the ASCII character set, including all upper and lower case letters, numerals, punctuation, and other control characters such as a line feed. A computer *word* is usually two, four, or eight bytes, and is the unit of storage that can be accessed simultaneously and moved back and forth from the central processing unit (CPU) to various storage devices. If we could manipulate bits directly, then we could represent each site in a Boolean cellular automaton by a bit and update a whole word of sites (32 sites on a 32 bit machine) simultaneously. This type of update is a simple example of parallel processing on a single processor machine.

The C language has intrinsic bit manipulation operations; most Fortran 77 and all Fortran 90 compilers also have intrinsic functions that perform bit manipulation. One of the enticements of cellular automata is that they can run very quickly on parallel processors. Special processing boards exist for personal computers and workstations that run cellular automata models faster than on a general purpose supercomputer. A major interest in cellular automata is how they can be used as a general paradigm for massively parallel computation (cf. Toffoli and Margolus).

## 15.2 Lattice Gas Models of Fluid Flow

One of the most promising applications of cellular automata models is to simulations of fluid flow. Fluid flow is very difficult to simulate because the partial differential equation describing fluid flow, the Navier-Stokes equation, is nonlinear. As we have found, nonlinear equations can lead to the breakdown of standard numerical algorithms. In addition, there are typically many length scales that must be considered simultaneously. These length scales include the microscopic motion of the fluid particles, the length scales associated with fluid structures such as vortices, and the length scales of macroscopic objects such as pipes or obstacles. Because of all these considerations, simulations of fluid flow based on direct numerical solutions of the Navier-Stokes equation typically require sophisticated numerical methods (cf. Oran and Boris).

The cellular automata models of fluids are known as *lattice gas* models. These models are based on the idea that if we maintain the fundamental conservation laws and symmetries associated with fluids, then we can produce the correct physics at the macroscopic level if we average over many particles. In a lattice gas model the positions of the particles are restricted to the sites of a lattice and the velocities are restricted to a small number of velocity vectors. A microscopic model needs to include two processes, the free motion between collisions and the collisions. In the simplest models, the particles move freely to their nearest neighbor lattice site in one time step. Then the velocities of the particles at each lattice site are updated according to a collision rule that conserves mass, momentum, and kinetic energy. The free motion and the collisions for all sites are computed simultaneously.

To understand the nature of lattice gas models, it is easier to discuss a specific model. Three-dimensional models are being studied, but they are much more difficult to visualize and to understand theoretically, and we will consider only two-dimensional models. We assume a triangular lattice, because its symmetry is more closely related to that of a continuum than a square lattice. In addition, collision rules for square lattices do not typically mix the horizontal and vertical motions of the particles. All particles are assumed to have the same speed and mass. The possible velocity vectors lie only along the links connecting sites, and hence there are only six possible velocities (labeled 0 to 5 because the first bit in a computer byte is in the 0 position):

$$\begin{array}{lll} \mathbf{v}_0 = (1, 0) & \mathbf{v}_1 = (1, -\sqrt{3})/2 & \mathbf{v}_2 = -(1, \sqrt{3})/2 \\ \mathbf{v}_3 = (-1, 0) & \mathbf{v}_4 = (-1, \sqrt{3})/2 & \mathbf{v}_5 = (1, \sqrt{3})/2 . \end{array}$$

In some models a rest particle also is allowed. Each site can have at most one particle moving in a particular direction. The update process proceeds by first moving all the particles in the direction of their velocity to a neighboring site. Then at each lattice site the velocity vectors are changed according to a collision rule. Particle number and kinetic energy are easily conserved because all particles have the same speed, and we need only insure momentum conservation. There

is some flexibility in the choice of rules. One set of collision rules is illustrated in Fig. 15.4. These rules are deterministic with only one possible set of velocities after a collision for each possible set of velocities before a collision.

Because True BASIC does not provide intrinsic bit manipulation functions, we do not list a program in True BASIC. Instead, we list a Fortran and C lattice gas program in Appendices ?? and ??, respectively. You can convert either program to True BASIC by using the fact that  $\text{mod}(\text{int}(A/2^{\text{xx}} \text{ caret } N), 2)$  is the  $n$ th bit of the integer  $A$ . We can determine if there is a particle at site  $x, y$  with velocity  $\mathbf{v}_2$  by testing whether  $\text{mod}(\text{int}(\text{lat}(x, y)/2^{\text{xx}} \text{ caret } N), 2)$  is equal to 1. Or it is possible to introduce a string of eight characters, each of which can be 0 or 1 as a model for a computer byte. For example, the  $n$ th lattice site with two particles of velocity  $\mathbf{v}_2$  and  $\mathbf{v}_4$  can be represented by the string array `lat$(n) = "001010"`. The string segment of `astring` between the  $p$ th and  $q$ th character is given by `astring$(p:q)`. We can determine if there is a particle with velocity  $\mathbf{v}_2$  by testing whether `lat$(n)[3:3]` is equal to "1".

We now describe the procedure for storing information about the particles in an array of integers, `lat`. The eight bits in a byte are labeled from 0 to 7. In principle, we could use a four byte integer to update four sites simultaneously, but to avoid complications we will not do so. Instead each site of the lattice is represented by one element of `lat`. We use the first six bits from 0 to 5 of an integer to represent particles moving in the six possible directions with bit 0 corresponding to a particle moving with velocity  $v_0$ . For example, if bit 0 equals unity, we know there is a particle at this site with velocity  $v_0$ . If there are three particles with velocities  $v_0, v_2$ , and  $v_4$  at a site, then this situation is represented by 00010101 in bit notation; the corresponding decimal equivalent is 21. From Fig. 15.4 we see that after the collision there are three particles with velocities  $v_1, v_3$ , and  $v_5$  corresponding to 42. We can express this collision as `rule(21) = 42`. Similar decimal equivalents can be expressed for all the other possible collisions.

We reserve bit 6 for a possible rest particle. If we want to occupy a site with a fixed particle to represent a barrier, we use bit 7, that is, we set the value of barrier sites equal to  $2^7 = 128$ . What boundary condition should we use when a particle is adjacent to a barrier site and heading toward it? The simplest rule that insures that we do not lose any particles is to set the velocity  $\mathbf{v}$  of such a particle equal to  $-\mathbf{v}$ . Other possibilities are to set the angle of incidence equal to the angle of reflection or to set the velocity to an arbitrary value. The latter case corresponds to a rough surface and is more difficult to implement because we need to insure that no site has more than one particle with the same velocity.

An important use of lattice gas models is to simulate flow in and around various geometries. The fluid velocity field can be seen to develop vortices, wakes, and other fluid structures near obstacles. Typically, particles are injected at one end and absorbed as they reach the other end. Periodic boundary conditions are used in the other direction. Very large lattices are required to obtain quantitative results, because it is necessary to average the velocity over many sites. The density of the fluid is determined by the average number of particles per site, and the pressure can be varied by changing the flux of particles that are injected at one end. We discuss some typical applications in Problems 15.6–15.8.

*Problem 15.6.* Approach to equilibrium

- a. To maintain zero total velocity or momentum at all times, use an initial configuration where all sites contain either no particles or six particles whose net momentum is zero. The number density  $\rho = 6N/(L_x L_y)$ , where  $N$  is the number of sites with six particles and the total number of

sites is  $L_x \times L_y$ . Use Program `latgas.f` in Appendix ?? or Program `latgas.c` in Appendix ?? as a basis for your program for simulating a lattice gas. Use periodic boundary conditions in both directions. The output of the program should be a pictorial representation of the average velocity at each site, for example, an arrow pointing in the direction of the average velocity with a size proportional to the magnitude of the average velocity. Begin with a dilute gas with  $N = 10$  on a  $10 \times 10$  triangular lattice. Plot the velocity vector field after each iteration of the lattice gas to check that your program is working correctly.

- b. We now consider the approach to equilibrium. Use a  $30 \times 30$  lattice and place six particles at every site in a  $4 \times 4$  region. Describe qualitatively what happens to the particles as a function of time. Approximately how many time steps does it take for the particles to fill the box? Do the particles appear to be at random positions with random velocities? Describe your visual algorithm for determining when equilibrium has been reached.
- c. Repeat part (b) for an initial  $b \times b$  region of particles with  $b = 2, 6, 8,$  and  $10$ . Estimate the equilibration time in each case. What is the qualitative dependence of the equilibration time on  $b$ ? How does the equilibration time depend on the number density  $\rho$ ?
- d. Repeat part (b) for an initial  $4 \times 4$  region of particles, but vary the size of the box. Try  $L_x = L_y = 10, 20,$  and  $40$ . Estimate the equilibration time in each case. How does the equilibration time depend on the number density  $\rho$ ?

*Problem 15.7.* Flow past a barrier

- a. Modify your program from Problem 15.6 so that at each time step three particles are injected with velocities  $v_0, v_1,$  and  $v_5$  at each site on the left-hand side. The particles are removed on the right-hand side. Include barrier sites in the middle of the cell representing a  $b_x \times b_y$  rectangular barrier. Use the barrier boundary condition that the directions of the reflected particles are reversed,  $\mathbf{v} \rightarrow -\mathbf{v}$ . Use periodic boundary conditions in the vertical direction. Besides the left-hand column and the barrier sites, all other sites are initially empty. Represent the velocity field visually as described in Problem 15.6a.
- b. Choose  $L_x = 50$  and  $L_y = 20$  with a barrier of dimensions  $b_x = 5$  and  $b_y = 1$ . Describe the flow once a steady state velocity field begins to appear. Can you see a wake appearing behind the obstacle? Are there vortices (circular fluid flow)?
- c. Repeat part (b) with different size obstacles. Are there any systematic trends?
- d. Reduce the pressure by injecting particles at the left every other time step. Are there any noticeable changes in behavior from parts (b) and (c)? Reduce the pressure still further and describe any changes in the fluid flow.
- e. Increase the size of the lattice by a factor of 10 in each direction and average the velocity in each  $5 \times 5$  region. Compare the flow patterns that you obtain with those obtained in parts (b)–(d).

*Problem 15.8.* Fluid flow in porous media

- a. Modify your lattice gas program so that instead of a rectangular barrier, the barrier sites are placed at random in the lattice. Add a subroutine that sums the horizontal velocity of those

particles that reach the right edge of the lattice. The current density is the average of this sum per unit height of the lattice. Compute the current density as a function of porosity, the fraction of sites not containing barriers. If time permits, average over at least ten pore configurations for each value of the porosity. Use  $L_x = 50$  and  $L_y = 20$ .

- b. Vary the size of the lattice and use the finite size scaling procedure discussed in Section 13.4 to estimate the critical exponent  $\mu$  defined by the dependence of the current density  $J$  on the porosity  $\phi$ . That is,  $J \sim (\phi - \phi_c)^\mu$ . Assume that you know the value of the percolation exponent  $\nu$  defined by the critical behavior of the connectedness length  $\xi \sim |p - p_c|^{-\nu}$  (see Table 13.1).

The principle virtues of lattice gas models are their use of simultaneous updating, which makes them very fast on parallel computers, and their use of integer or boolean arithmetic, which might be faster than floating point arithmetic. Their major limitation is that many sites must be averaged over to obtain quantitative results. It is not yet clear whether lattice gas models are more efficient than standard simulations of the Navier-Stokes equation. The greatest promise for lattice gas models may not be with simple single component fluids, but with multicomponent fluids such as binary fluids and fluids containing bubbles.

### 15.3 Self-Organized Critical Phenomenon

In nature we rarely see very large events such as a magnitude eight earthquake, an avalanche on a snow covered mountain, the sudden collapse of an empire (for example, the Soviet Union), or the crash of the stock market. When such rare events occur, are they due to some special set of circumstances or are they a part of a more general pattern of events that would occur without any specific external intervention? The idea of *self-organized criticality* is that in many cases very large events are part of a distribution of events and do not depend on special conditions or external forces.

If  $s$  represents the magnitude of an event, such as the energy released in an earthquake or the amount of snow in an avalanche, then a system is said to be *critical* if the number of events  $N(s)$  follows a power law:

$$N(s) \sim s^{-\alpha}. \text{(no characteristic scale)} \quad (15.4)$$

One implication of the form (15.4) is that there is no characteristic scale. Systems whose correlation or distribution functions decay as power laws are said to be *scale invariant*. This terminology reflects the fact that power laws look the same on all scales. For example, the replacement  $s \rightarrow bs$  in the function  $N(s) = As^{-\alpha}$  yields a function  $\tilde{N}(s)$  that is indistinguishable from  $N(s)$ , except for a change in the amplitude  $A$  by the factor  $b^{-\alpha}$ . If  $\alpha \approx 1$ , there would be one large event of size 1000 for every 1000 events of size one.

Contrast the power law dependence of  $N(s)$  in (15.4) to the result of combining a large number of independently acting random events. In this case we know that the distribution of the sum is a Gaussian (see Problem 12.8) and that  $N(s)$  has the form

$$N(s) \sim e^{-(s/s_0)^2}. \text{(characteristic scale)} \quad (15.5)$$

Scale invariance does not hold for functions that decay exponentially, because the replacement  $s \rightarrow bs$  in the function  $e^{-(s/s_0)^2}$  changes  $s_0$  (the characteristic scale of  $s$ ) by the factor  $b^2$ . We note that for a power law distribution, there are events of all sizes, but for a Gaussian distribution, there are practically speaking no large events.

A common example of self-organized critical phenomena is an idealized sand pile. Suppose that we construct a sand pile by randomly adding one grain at a time onto a flat surface with open edges. Initially, the grains will stay more or less where they land, but after a while there will be small avalanches during which the grains move so that the local slope of the pile is not too large. Eventually, the pile reaches a statistically stationary (time-independent) state and the amount of sand added balances the sand that falls off the edge (on the average). When a single grain of sand is added to a configuration belonging to this state, a rearrangement might occur that triggers an avalanche of any size (up to the size of the system), so that the mean slope again equals the critical value. We say that the statistically stationary state is critical because there are avalanches of all sizes. The stationary state is self-organized because no external parameter (such as the temperature) needs to be tuned to force the system to this state. In contrast, the concentration of fissionable material in a nuclear chain reaction has to be carefully controlled for the nuclear chain reaction to be exactly critical.

The nature of self-organized critical phenomena can be understood by considering some simple models. We begin with a one-dimensional model based on the simplified behavior of a sand pile. Consider a lattice of  $L$  sites and let the height at each site be represented by the array element  $h(i)$ . One grain of sand is added to the left-most site,  $h(1) = h(1) + 1$ , at each time step. During this time step all the sites are checked to see if  $h(i) - h(i+1) > 1$ . All sites that satisfy this condition are marked for “toppling.” Next each marked site is toppled. A simple rule is to let  $h(i) = h(i) - 1$  and  $h(i+1) = h(i) + 1$ , that is, the sand falls to the right. Any grains of sand that go beyond  $i = L$  are lost forever. Program `sandpile` implements this simple model. We will find in Problem 15.9 that slightly more complicated rules are necessary to find nontrivial behavior.

```
PROGRAM sandpile
! simple one-dimensional sandpile model
DIM height(51),move(100),N(0:51)
CALL initial(height(),N(),L,sand$,#1,#2)
LET grain = 0
DO
  LET height(1) = height(1) + 1 ! add grain to first site
  WINDOW #1
  BOX SHOW sand$ at 1,height(1) - 1
  LET topple = 0 ! number of grains that topple
  LET grain = grain + 1 ! number of grains added
  DO
    CALL check(height(),L,move(),unstable)
    IF unstable = 1 then
      CALL slide(height(),L,move(),sand$,#1)
      LET topple = topple + 1
    END IF
  LOOP until unstable = 0
```

```

    LET N(topple) = N(topple) + 1
    CALL show_distribution(N(),L,grain,#2)
LOOP until key input
END

SUB initial(height(),N(),L,sand$,#1,#2)
OPEN #1: screen 0,0.7,0,1
INPUT prompt "lattice size L = ": L      ! suggest L = 20
FOR i = 1 to L
    LET height(i) = 0
NEXT i
LET height(L+1) = 0          ! boundary condition
SET WINDOW -1,L+1,-2,40
SET COLOR "blue"
BOX AREA 1.1,1.9,0.1,0.9
BOX KEEP 1,2,0,1 in sand$
CLEAR
SET COLOR "black"
PLOT LINES: 1,-0.05;L+1,-0.05
! initialize array
FOR topple = 0 to 20
    LET N(topple) = 0
NEXT topple
OPEN #2: screen 0.72,1,0.4,1
SET WINDOW -0.1,L+1,-0.05,1.01
PLOT LINES: 0,-0.01;L+0.2,-0.01
PLOT LINES: -0.05,-0.01;-0.05,1
END SUB

SUB check(height(),L,move(),unstable)
LET unstable = 0
FOR i = 1 to L
    IF height(i) - height(i+1) > 1 then
        LET move(i) = 1
        LET unstable = 1
    ELSE
        LET move(i) = 0
    END IF
NEXT i
END SUB

SUB slide(height(),L,move(),sand$,#1)
WINDOW #1
FOR i = 1 to L
    IF move(i) = 1 then
        LET height(i) = height(i) - 1    ! sand topples
    
```

```

        BOX CLEAR i,i + 1,height(i),height(i) + 1
        IF i < L then
            ! next site receives grain
            LET height(i+1) = height(i+1) + 1
            BOX SHOW sand$ at i+1,height(i+1) - 1
        END IF
    END IF
NEXT i
END SUB

SUB show_distribution(N(),L,grain,#2)
WINDOW #2
! erase previous graph
SET COLOR "white"
BOX AREA 0,L+0.2,0,1
SET COLOR "black"
FOR topple = 0 to L
    IF N(topple) > 0 then
        BOX AREA topple,topple+0.2,0,N(topple)/grain
    END IF
NEXT topple
END SUB

```

*Problem 15.9.* One-dimensional sand piles

- a. Use Program `sandpile` with  $L = 20$ . Where is the sand added? What is the slope of the sand pile after a long time?
- b. Program `sandpile` computes the number of sites  $s$  that topple during each time step and plots the distribution of toppling sizes,  $N(s)$ , versus  $s$ . Modify the program so that  $N(s)$  is computed only after the sand pile reaches a steady state. Is the behavior of  $N(s)$  interesting for this model?
- c. Modify the rules so that a site which topples loses *two* grains of sand, one to its nearest neighbor to the right and the other to its next nearest neighbor to the right. Plot the distribution  $N(s)$  versus  $s$  after steady state behavior is reached. Is there a range of values of  $s$  for which  $N(s)$  shows power law behavior? If so, make the appropriate plot and estimate the critical exponent  $\alpha$ .
- d. Introduce the variable  $m(i) = h(i + 1) - h(i)$ , and convince yourself that the same results are obtained by replacing  $h(i)$  by  $m(i)$  and using the rule  $m(i) > 1$  for toppling. The variable  $m(i)$  is called the local slope. Modify your program so that  $m(i)$  is used instead of  $h(i)$  and adopt the toppling rule used in part (c). Do your results change if you add a grain of sand at each time step at random anywhere in the lattice?
- e. Use the same rule that you considered in parts (c) and (d) and compute  $N(s)$  for different values of  $L$  and systematically investigate the importance of finite size effects. Average  $N(s)$  over many updates and obtain your best estimate for the critical exponent  $\alpha$ .

In Problem 15.9 we saw that the fundamental variable is the local slope. Most sand pile models are described using this variable. In the literature many authors refer to the height when they really mean the local slope. In Problem 15.10 we explore the behavior of a two-dimensional model of a sand pile.

*Problem 15.10.* Two-dimensional sand pile

- a. Write a program to simulate a two-dimensional sand pile using the rule that a site topples if  $m(i) > 3$ . The pile is grown by choosing a site at random and adding one grain, that is,  $m(i) \rightarrow m(i) + 1$ . If site  $i$  topples (exceeds its critical value), it distributes four grains of sand to its four nearest neighbors (on a square lattice), that is,  $m(i) \rightarrow m(i) - 4$ . At the edges or corners, only three or two neighbors, respectively, are affected, and the sand that goes outside the boundary of the lattice is lost. Color code the sites according to their value of  $m(i)$ . In Program `sandpile` we checked all sites for stability, but in two dimensions such a check would take too much time if the size of the lattice were sufficiently large. For this reason we suggest that you maintain two arrays `posx` and `posy` which contain the  $x$  and  $y$  coordinates of those sites that need to be checked for stability. Each time a site topples, add its neighbors' positions to these arrays. Then remove the last site from the arrays and check its stability. Continue this process of removing sites and checking their stability, and adding neighbors to the array until there are no more sites to check.
- b. After the critical state has been reached, compute the number of sites  $s$  that topple in response to the addition of a single grain. Then compute the distribution of toppling sizes  $N(s)$  and determine if  $N(s)$  exhibits power law behavior. Begin with  $L = 10$  and then consider larger values of  $L$ .
- c. Although the model in part (a) might seem reasonably realistic, it is of course over simplified. Laboratory experiments indicate that real sand piles show power law behavior if the piles are small, but larger sand piles do not (see Jaeger et al.). Modify the model to make its behavior more realistic.

Do model sand piles have anything in common with earthquakes? The Gutenberg-Richter law for  $N(E)$ , the number of earthquakes with energy release  $E$ , has been observed empirically and is consistent with power law behavior:

$$N(E) \sim E^{-b}, \quad (15.6)$$

with  $b \approx 0.5$ . The magnitude of earthquakes on the Richter scale is approximately the logarithm of the energy release. One implication of the power law dependence in (15.6) is that there is nothing special about large earthquakes. That is, if we could wait a couple of million years, we would likely observe earthquakes of size ten following the same Gutenberg-Richter law. In Problem 15.11 we explore the behavior of a model of tectonic plate motion which suggests that the Gutenberg-Richter law is a consequence of self-organized criticality.

*Problem 15.11.* Earthquake model

Given the long time scales between earthquakes and the complexity of the historical record, there is considerable interest in developing ways of studying earthquakes using simulations. The Burridge and Knopoff model of fault systems consists of a system of coupled masses in contact with a moving

rough surface. The masses are subjected to static and dynamic frictional forces, and also are pulled by an external force corresponding to slow tectonic plate motion. The major difficulty with this model and similar ones that have been proposed is that the numerical solution of the corresponding Newton's equations of motion is computationally intensive. For this reason we first study a simple cellular automaton model that retains some of the basic physics of the Burridge and Knopoff type models. A more realistic cellular automaton model is discussed in Project 15.19.

Define a real variable  $F(i, j)$  on a square lattice, where  $F$  represents the force on the block at position  $(i, j)$ . The linear dimension of the lattice is  $L$ , and the number of sites  $N$  is given by  $N = L^2$ . The initial state of the lattice at time  $t = 0$  is found by assigning small random values to  $F(i, j)$ . The lattice is updated according to the following rules:

1. Increase  $F$  everywhere by a small amount  $\Delta F$ , for example, choose  $\Delta F = 10^{-5}$ , and increase the time  $t$  by 1. This increase represents the effect of the driving force due to the slow motion of the tectonic plate.
2. Check if  $F(i, j)$  is greater than  $F_c$ , the critical threshold value of the force. If not, the system is stable and step 1 is repeated. If the system is unstable, go to step 3. Choose  $F_c = 4$  for convenience.
3. The release of force due to slippage of a block is represented by letting  $F(i, j) = F(i, j) - F_c$ . The transfer of force is represented by updating the force at the sites of the four neighbors at  $(i, j \pm 1)$  and  $(i \pm 1, j)$ :  $F \rightarrow F + 1$ .

As an example, let the linear dimension of the lattice  $L = 10$  so that the number of sites  $N = L^2 = 100$ . Do the simulation and show that the system eventually comes to a statistically stationary state, where the average value of the force at each site stops growing. Monitor the distribution of the size  $s$ , where  $s$  is the total number of sites (blocks) that are affected by an instability. Increase  $L$  to  $L = 30$  and repeat your simulations.

The behavior of some other simple models is explored in various contexts in the following four problems.

*Problem 15.12.* Forest fire model

- a. Consider the following simple model of the spread of a forest fire. Suppose that at  $t = 0$  the  $L \times L$  sites of a square lattice either have a tree or are empty with probability  $p_t$  and  $1 - p_t$  respectively. Those sites which have a tree, are on fire with probability  $f$ . At each time step an empty site grows a tree with probability  $p$ , a tree that has a nearest neighbor site on fire catches fire, and a site that is already on fire dies and becomes empty. Note that the changes in each site occur synchronously, and that this model is an example of a probabilistic cellular automaton. Write a program to simulate this model and color code the three types of sites. Use periodic boundary conditions.
- b. Use  $L \geq 30$  and determine the values of  $p$  for which the forest maintains fires indefinitely. Note that as long as  $p > 0$ , new trees will always grow.
- c. Choose the value of  $p$  that you found in part (b) and compute the distribution of the number of sites  $s_f$  on fire. If the distribution is critical, determine the exponent  $\alpha$  that characterizes this

distribution. Also compute the distribution for the number of trees,  $s_t$ . Is there any relation between these two distributions?

- d. To obtain reliable results it is frequently necessary to average over many initial configurations. However, it is possible that the behavior of a system is independent of the initial configuration and averaging over many initial configurations is unnecessary. This latter possibility is called *self-averaging*. Repeat parts (b) and (c), but average your results over ten initial configurations. Is this forest fire model self-averaging?

*Problem 15.13.* Another forest fire model

- a. Consider a simple variation of the model discussed in Problem 15.12. At  $t = 0$  each site is occupied by a tree with probability  $p_t$ ; otherwise, it is empty. The system is updated in successive time steps as follows:
- (a) Randomly grow new trees at time  $t$  with a small probability  $p$  from sites that are empty at time  $t - 1$ ;
  - (b) A tree that is not on fire at  $t - 1$  catches fire due to lightning with probability  $f$ ;
  - (c) Trees on fire ignite neighboring trees, which in turn ignite their neighboring trees, etc. The spreading of the fire occurs instantaneously.
  - (d) Trees on fire at time  $t - 1$  die (become empty sites) and are removed at time  $t$  (after they have set their neighbors on fire);

As in Problem 15.12, the changes in each site occur synchronously. Determine  $N(s)$ , the number of clusters of trees of size  $s$  that catch fire in each time step. Two trees are in the same cluster if they are nearest neighbors.

- b. Do the simulation and determine if the behavior of  $N(s)$  is consistent with  $N(s) \sim s^{-\alpha}$ . If so, estimate the exponent  $\alpha$  for several values of  $p$  and  $f$ .
- c. The balance between the mean rate of birth and burning of trees in the steady state suggests a value for the ratio  $f/p$  at which this model is likely to be scale invariant. If the average steady state density of trees is  $\rho$ , then at each time step the mean number of new trees appearing is  $pN(1 - \rho)$ , where  $N = L^2$  is the total number of sites. In the same spirit, we can say that for small  $f$ , the mean number of trees destroyed by lightning is  $f\rho N\langle s \rangle$ , where  $\langle s \rangle$  is the mean number of trees in a cluster. Is this reasoning consistent with the results of your simulation? If we equate these two rates, we find that  $\langle s \rangle \sim ((1 - \rho)/\rho)(p/f)$ . Because  $0 < \rho < 1$ , it follows that  $\langle s \rangle \rightarrow \infty$  in the limit  $f/p \rightarrow 0$ . Given the relation  $\langle s \rangle = \sum_{s=1}^{\infty} sN(s)/\sum_s N(s)$  and the divergent behavior of  $\langle s \rangle$ , why does it follow that  $N(s)$  must decay more slowly than exponentially with  $s$ ? This reasoning suggests that  $N(s) \sim s^{-\alpha}$  with  $\alpha < 2$ . Is this expectation consistent with the results that you obtained in part (b)?

In this model there are three well separated time scales, that is, the time for lightning to strike ( $\propto f^{-1}$ ), the time for trees to grow ( $\propto p^{-1}$ ), and the instantaneous spreading of fire through a connected cluster. This separation of time scales seems to be an essential ingredient for self-organized criticality (cf. Grinstein and Jayaprakash).

\**Problem 15.14.* Is “Life” critical?

Despite the simplicity of the rules of the Game of Life (see Problem ??), the dynamics of the game are not well understood. For example, if each cell is randomly populated at  $t = 0$ , how does the system develop structure and how can we characterize it? The existence of self-organized criticality in various cellular automata models has led several workers to investigate if similar phenomena exist in Life. The results are not yet definitive, and this problem will require considerable computational power and most likely some insight before it is solved. The idea is to begin with a random distribution ( $p = 0.5$ ) of live cells and let the system evolve until only static or simple local periodic activity is found. Then a single dead cell is chosen at random and changed to a live cell. This change is analogous to adding a grain of sand to the sand pile model. The system is allowed to evolve until it again reaches a stable or periodic configuration. The quantities of interest include  $s$ , the total number of sites that are changed after the initial change, and  $T$ , the number of updates that are needed to return to a stable or periodic configuration. Then choose another dead cell at random, and repeat the above procedure. Compute  $N(s)$  and  $D(T)$ , the distribution of  $s$  and  $T$ , respectively. If a site is changed several times after a perturbation, each change counts as part of the response. The difficult part of the program is determining whether a configuration is part of a periodic cycle. For small periods the lattice can be stored for a few times and then compared to previous configurations to check whether the state has repeated itself. It is very difficult to check for all types of periodic states, but if the system is started from a random distribution of live sites, cyclic structures with long periods are very rare. Another way of improving the efficiency is to change a dead cell to a live cell only if there is at least one live cell in its neighborhood (for example, the cell’s twenty nearest neighbors). In this way we do not waste time counting small avalanches, because if there are very few live cells in a neighborhood, then usually the system will return to a stable or periodic state very quickly.

Consider at least a  $50 \times 50$  lattice with periodic boundary conditions. Are your results for  $N(s)$  and  $D(T)$  consistent with power law behavior? Consider progressively larger lattices and compute the mean values  $\langle s \rangle$  and  $\langle T \rangle$  in addition to  $N(s)$  and  $D(T)$ . If  $N(s)$  and  $D(T)$  exhibit critical behavior, then the corresponding mean values would increase with the size of the lattice. Why? At present, some workers believe that  $N(s)$  and  $D(T)$  exhibit critical behavior, while others believe that this apparent behavior is an artifact resulting from the relatively small lattices that have been considered. (The largest lattices considered at present are  $1024^2$ .) Can you reach any tentative conclusions on the basis of your results?

*Problem 15.15.* Model of punctuated equilibrium

- a. The idea of *punctuated equilibrium* is that biological evolution occurs episodically rather than as a steady, gradual process. That is, most of the major changes in life forms occur in relatively short periods of time. Bak and Sneppen have proposed a simple model that exhibits some of the dynamical behavior expected of punctuated equilibrium. The model consists of a one-dimensional cellular automata of size  $L$ , where cell  $i$  represents the biological fitness of species  $i$ , normalized to unity. Initially, all cells receive a random fitness  $f_i$  between 0 and 1. Then the cell with the lowest fitness and its two nearest neighbors are randomly given new fitness values. This update rule is repeated indefinitely. Write a program to simulate the behavior of this model. Use periodic boundary conditions, and show the fitness of each cell as a bar of height  $f_i$ .
- b. Begin with  $L = 64$  and describe what happens to the distribution of fitness values after a long

time. We can crudely think of the update process as replacing a species and its neighbors by three new species. In this sense the fitness represents a barrier to creating a new species. If the barrier is low, it is easier to create a new species. Do the low fitness species die out? What is the average value of fitness of the species after the model is run for a long time ( $10^3$ ,  $10^4$ , or more time steps)? Compute the distribution of fitness values,  $N(f)$ , averaged over all cells and over a long time. Allow the system to come to a fluctuating steady state before computing  $N(f)$ . Plot  $N(f)$  versus  $f$ . Is there a critical value  $f_c$  below which  $N(f)$  is much less than the values above  $f_c$ ? Is the update rule reasonable from an evolutionary point of view?

- c. Modify your program to compute the distance  $x$  between successive fitness changes and the distribution of these distances  $C(x)$ . Make a log-log plot of  $C(x)$  versus  $x$ . Is there any evidence of self-organized criticality?
- d. Another way to visualize the results is to make a plot of the time at which a cell changed versus the position of the cell. Is the distribution of the plotted points approximately uniform? We might expect that the time of survival of a species depends exponentially on its fitness, and hence each update corresponds to an elapsed time of  $e^{-cf_i}$ , where the constant  $c$  sets the time scale and  $f_i$  is the fitness of the cell which has been changed. Choose  $c = 100$  for convenience and make a similar plot with the time axis replaced by the logarithm of the time, that is, the quantity  $100f_i$ . Is this plot more meaningful?
- e. Another way of visualizing the meaning of punctuated equilibrium is to plot the number of times groups of cells change as a function of time. Divide the time into units of 100 updates and compute the number of fitness changes for cells  $i = 1$  to 10 as a function of time. Do you see any evidence of punctuated equilibrium?

Stuart Kauffman has devised a cellular automaton model of genetics in which each site interacts with  $K$  neighbors through a randomly selected cellular automata rule. The  $K$  neighbors of each site also are randomly chosen at the beginning of a run. For  $K = 4$ , there are  $2^4 = 16$  neighbor configurations and  $2^{16} = 65536$  possible rules. In the Kauffman model each site is initially assigned one of the 65536 possible rules (for  $K = 4$ ). After some time the system goes either to a limit cycle (repeating itself after a finite number of states) or to a fixed point (the state does not change). The idea is to test if a change of the update rule of a single site drives the entire system out of its steady state, that is, if a small mutation drastically changes the genetic behavior. For  $K = 2$ , simulations show that the genetic behavior withstands the mutation. However, for large  $K$ , for example,  $K = 10$ , a single mutation leads to a very different state. Perhaps, Nature is an example of a complex system where small local changes can lead to large global changes.

## 15.4 Neural Networks

Can computers think? This question has occupied philosophers, computer scientists, cognitive psychologists, and many others ever since the first computer was imagined. The assumption of workers in “strong” *artificial intelligence* is that it will be possible someday for computers to think. The reasoning is that thinking is based on symbolic manipulation of inputs from the external world. Of course, everyone agrees that we are far from having a computer think like a human. Some contend that computers will be able to only simulate the human brain and not be able to think

like it. Part of the argument is that the brain is not analogous to the hardware of a computer, and the mind is not analogous to its software.

Recent developments in two classes of models, known as neural networks and genetic algorithms, tend to weaken one argument against AI. These models are based on the idea that the program can change itself based on the inputs, that is, the program statements and its data are the “mind” of the computer, and the program and its data can change depending on its own internal state and outside inputs. Indeed, we should consider the entire memory of the computer to constitute its mind. The result is that the state of the computer can change itself in ways that the programmer cannot anticipate. As we have learned, simple algorithms can lead to complex and unpredictable outcomes, and it probably comes as no surprise that the state of the computer can evolve through a sequence of states that can be very complex, that is, neither completely random nor completely ordered. Perhaps, the mind exists at the edge of chaos where the complex behavior just begins.

Neural networks model a piece of this ultimate computer mind. The idea is to store memories so that a computer can recall them when inputs are given that are close to a particular memory. As humans we have our own algorithms for doing so. For example, if we see someone more than once, the person’s face might provide input that helps us recall the person’s name. In the same spirit, a neural network can be given a pattern, for example, a string of 0’s and 1’s, that partially reflect a previously memorized pattern. The network then attempts to recall the memorized pattern. The significant difference between what the computer usually does to retrieve data from its memory and the memory recall of a neural network is that in the latter we consider *content addressable memory* in contrast to computer programs themselves which retrieve memory based on the address or location of the data, not on its content.

Neural network models have been motivated by how neurons in the brain might collectively store and recall memories. It is known that a neuron “fires” once it receives electrical inputs from other neurons whose strength reaches a certain threshold. An important characteristic of a neuron is that its output is a nonlinear function of the sum of its inputs. Usually, a neuron is in one of two states, a resting potential (not firing) or firing at the maximum rate. The assumption is that when memories are stored in the brain, the strengths of the connections between neurons change. Neural network models attempt to maintain the key functions of biological neurons without the specific biological substrate.

We now consider an example of a neural network due to Hopfield. The network consists of  $N$  neurons and the state of the network is defined by the potential at each neuron,  $V_i$ . The strength of the connection between the  $i$ th and  $j$ th neuron is denoted by  $T_{ij}$  and is determined by the  $M$  stored memories:

$$T_{ij} = \sum_{s=1}^M (2V_i^s - 1)(2V_j^s - 1). \quad (15.7)$$

$V^s$  is the state of the  $s$ th stored memory. Given an initial state  $V_i^0$ , the dynamics of the network is simple, that is, choose a neuron  $i$  at random and change its state according to its input. Its input strength,  $S_i$ , is defined as

$$S_i = \sum_{j \neq i} T_{ij} V_j, \quad (15.8)$$

where  $V_j$  represent the current state of the  $j$ th neuron. Change the state of neuron  $i$  by setting

$$V_i = \begin{cases} 1, & \text{if } S_i > 0 \\ 0, & S_i \leq 0. \end{cases} \quad (15.9)$$

Note that the threshold value has been set equal to zero, but other values could be used as well.

Program `hopfield`, listed in the following, implements this model of a neural network and stores memories and recalls them based on user input.

```

PROGRAM hopfield
DIM T(50,50)
CALL memorize(T(,),N)
DO
  CALL recall(T(,),N)
LOOP
END

SUB memorize(T(,),N)
  DIM V(50)
  RANDOMIZE
  INPUT prompt "number of stored memories = ": M
  ! N corresponds to number of neurons
  INPUT prompt "size of memories = ": N
  PRINT "enter M strings of N 0's and 1's"
  FOR memory = 1 to M
    LINE INPUT s$
    CALL convert(s$,N,V())
    FOR i = 1 to N
      FOR j = 1 to N
        IF i <> j then
          LET T(i,j) = T(i,j) + (2*V(i) - 1)*(2*V(j) - 1)
        END IF
      NEXT j
    NEXT i
  NEXT memory
  PRINT
END SUB

SUB recall(T(,),N)
  DIM V(50)
  PRINT "enter a string of N 0's and 1's"
  LINE INPUT s$
  CALL convert(s$,N,V())
  DO
    FOR k = 1 to N
      LET i = int(N*rnd) + 1      ! choose neuron at random

```

```

        LET sum = 0
        FOR j = 1 to N
            IF i <> j then LET sum = sum + T(i,j)*V(j)
        NEXT j
        IF sum > 0 then
            LET V(i) = 1           ! above threshold
        ELSE
            LET V(i) = 0           ! below threshold
        END IF
    NEXT k
    FOR i = 1 to N
        PRINT using "#": V(i);
    NEXT i
    PRINT
    PAUSE 1
    LOOP until key input
    GET KEY kk
END SUB

SUB convert(s$,N,V())
    ! convert string to array of 0's and 1's
    FOR i = 1 to N
        LET c$ = s$[i:i]
        IF c$ = "0" then
            LET V(i) = 0
        ELSE
            LET V(i) = 1
        END IF
    NEXT i
END SUB

```

*Problem 15.16.* Memory recall in the Hopfield model

- a. Use Program `hopfield` to explore the ability of the Hopfield neural network to store and recall memories. Begin by storing  $M = 2$  memories of  $N = 20$  characters each. For example, store 11111000000000011111 and 11001100110011001100. Then try to recall a memory using the input string 11111110000001111111. This input is similar to the first memory. Record the “Hamming” distance between the final state and the closest memory, where the Hamming distance is the number of characters that differ between two strings. Repeat the above procedure for a number of different values of the number of memories  $M$  and the memory length  $N$ .
- b. Estimate how many memories can be stored for a given sized string before recall becomes severely reduced. Make estimates for  $N = 10, 20$ , and  $30$ . What criteria did you adopt for correct recall?
- c. Modify Program `hopfield` so that two-dimensional patterns can be memorized. Instead of a string of 1's and 0's, you will need a grid of  $L \times L$  cells each of which can be on or off. The

major change in your program is that the indices  $i$  and  $j$  for  $S_i$  and  $T_{ij}$  must now refer to a particular cell. For example, if  $(x, y)$  is the location of the  $i$ th cell, then  $i = x + (y - 1)L$ .  $T_{ij}$  will be represented by an  $L^2 \times L^2$  array, and  $V_i$  will correspond to an array of length  $L^2$ . Also, you will need to write input and output subroutines to show the patterns. Consider a grid with  $L \geq 10$  and store three patterns. Patterns could be simple geometric shapes or symbols. Then input a pattern similar to one of the stored memories and see how well the Hopfield algorithm is able to recall the correct pattern. Repeat for a number of different input patterns, and then increase the number of stored memories.

In addition to helping us understand biological neural networks, neural networks can be used to determine an optimal solution to a difficult optimization problem. In Problem 15.17 we consider the problem of finding the minimum energy of a model spin glass.

*Problem 15.17.* Minimum energy of an Ising spin glass

- a. We can define an energy for the Hopfield model in analogy to the Ising model:

$$E = -\frac{1}{2} \sum_i \sum_{j \neq i} T_{ij} V_i V_j, \quad (15.10)$$

where we assume that  $T_{ij} = T_{ji}$ . Note that the form of (15.10) is very similar to (15.2). If we give the  $T_{ij}$  random values, then the model is an example of a “spin glass” with long-range interactions (see Project ??). Modify your program so that the  $T_{ij}$  are given random values between  $-1$  and  $1$  with  $T_{ii} = 0$ . The program should ask for an input string of  $N$  characters. Have the program display the output string and the energy after every  $N$  attempts to change a neuron. Begin with  $N = 20$ .

- b. Describe what happens to the energy after a long time. For different initial states, but the same set of  $T_{ij}$ , is the energy the same after the system has evolved for a long time? Explain your results in terms of the number of local energy minima.
- c. What is the behavior of the states? Do you find periodic behavior and/or random behavior or do the states evolve to a state that does not change?

## 15.5 Genetic Algorithms

There are many people who find it difficult to accept that evolution is sufficiently powerful to generate the biological complexity seen in nature. Part of this difficulty arises from the inability of humans to intuitively grasp time scales that are much greater than their own lifetimes. Another reason is that it is very difficult to appreciate how random changes can lead to emergent complex structures. Genetic algorithms provide one way of understanding the nature of evolution. Their principal utility at present is in optimization problems, but they also are being used to model biological and social evolution. One of the important examples is due to the biologist Tom Ray (see Lewin) who used a genetic algorithm in conjunction with low level machine coding. The memory of the computer was loaded with code segments which reproduce with small changes in their code. Because each code segment requires memory and the computer’s processing time, the code segments compete with one another for memory and time. In what was a surprise to many,

the memory of the computer, which initially contained a few simple code segments, evolved into a complicated “ecosystem” of code segments of many different sizes and structures.

The idea of genetic algorithms is to model the process of evolution by natural selection. This process involves two steps: random changes in the genetic code during reproduction, and selection according to some fitness criteria. In biological organisms the genetic code is stored in the DNA. We will store the genetic code as a string of 0’s and 1’s. (In Fortran and C the genetic code can be stored as an integer variable and bit manipulation can be used.) The genetic code constitutes the *genotype*. The conversion of this string to the organism or *phenotype* depends on the problem. The selection criteria is applied to the phenotype. First we describe how change is introduced into the genotype.

Typically, nature changes the genetic code in two ways. The most obvious, but least often used method, is *mutation*. Mutation corresponds to changing a character at random in the genetic code string from 0 to 1 or from 1 to 0. The second and much more powerful method is associated with sexual reproduction. We can take two strings, remove a piece from one string and exchange it with the same length piece from the other string. For example, if string  $A = 0011001010$  and string  $B = 0001110001$ , then exchanging the piece from position 4 to position 7 leads to two new strings  $A' = 0011110010$  and  $B' = 0001001001$ . This type of change is called *recombination* or *crossover*. At each generation we produce changes using recombination and mutation. We then select from the enlarged population of strings (including strings from the previous generation), a new population for the next generation. Usually, a constant population size is maintained from one generation of strings to the next.

We next have to choose a selection criterion. If we want to model an actual ecosystem, we can include a physical environment and other sets of populations corresponding to different species. The fitness could depend on the interaction of the different species with one another, the interaction within each species, and the interaction with the physical environment. In addition, the behavior of the populations might change the environment from one generation to the next. For simplicity, we will introduce the idea of genetic algorithms with a single population of strings, a simple phenotype, and a simple criteria for fitness.

The phenotype we consider is a variant of the Ising spins considered in Problems 15.5 and 15.17. We consider a square lattice of linear dimension  $L$  occupied by spins that have one of the two values  $s_i = \pm 1$ . The energy of the system is given by

$$E = - \sum_{i,j=\text{nn}(i)} T_{ij} s_i s_j, \quad (15.11)$$

where the sum is over all pairs of spins that are nearest neighbors. Note that the energy function in (15.11) assumes that only nearest neighbor spins interact, in contrast to the energy function in (15.10) which assumes that every spin interacts with every other spin. The coupling constants  $T_{ij}$  can be either +1 (the ferromagnetic Ising model), -1 (the antiferromagnetic Ising model), randomly distributed (a spin glass), or have some other distribution. We adopt the energy as a measure of fitness. If we assume that  $|T_{ij}| = 1$ , then the minimum energy equals  $-2L^2$  and the maximum energy is  $2L^2$ . More precisely, we choose the combination  $2L^2 - E$ , a quantity that is always positive, as our measure of fitness, and take the probability of selecting a particular string with energy  $E$  for the next generation to be proportional to the fitness  $2L^2 - E$ .

How does a genotype become “expressed” as a phenotype? The genotypes consists of a list or

string of length  $L^2$  with 1's and 0's. Lattice site  $(i, j)$  corresponds to the  $n$ th position in the string where  $n = (j - 1)L + i$ . If the character in the string at position  $n$  is 0, then the spin at site  $(i, j)$  equals  $-1$ . If the character is 1, then the spin equals  $+1$ .

We now have all the ingredients we need to apply the genetic algorithm. Program `genetic` chooses a random size and a random position for the process of recombination, and periodic boundary conditions are applied to the string for the purpose of recombination. We use the True BASIC concatenation operator, `&`, to piece two strings together. For example, `"Magic" & "and" & "Larry"` is equivalent to `"MagicandLarry"`. Note that the selection of the population for the new generation uses the method of discrete nonuniform probability distributions discussed in Section 11.5.

```

PROGRAM genetic
DIM s$(1000),T(400,2),Eselect(1000)
CALL initial(s$( ),L,L2,T( ),npop,nrecombine,nmutation,ngeneration)
FOR igeneration = 1 to ngeneration
  LET ntot = npop
  FOR iswap = 1 to nrecombine
    CALL recombine(s$( ),L2,npop,ntot)
  NEXT iswap
  FOR i = 1 to nmutation
    CALL mutate(s$( ),L2,npop,ntot)
  NEXT i
  CALL selection(s$( ),L,L2,T( ),npop,ntot,Eselect( ))
NEXT igeneration
CALL showoutput(s$( ),npop,Eselect( ))
END

SUB initial(s$( ),L,L2,T( ),npop,nrecombine,nmutation,ngeneration)
RANDOMIZE
INPUT prompt "string (lattice) size = ": L
LET L2 = L*L
INPUT prompt "population number = ": npop
! L is linear dimension of phenotype
INPUT prompt "number of recombinations per generation = ": nrecombine
INPUT prompt "number of mutations per generation = ": nmutation
INPUT prompt "number of generations = ": ngeneration
! create random population of genotypes
FOR ipop = 1 to npop
  LET s$(ipop) = ""
  FOR i = 1 to L2
    IF rnd > 0.5 then
      LET s$(ipop) = s$(ipop) & "1"
    ELSE
      LET s$(ipop) = s$(ipop) & "0"
    END IF
  NEXT i
NEXT ipop

```

```

NEXT ipop
! create random bond network of Tij's
FOR i = 1 to L2
  FOR j = 1 to 2
    IF rnd > 0.5 then
      LET T(i,j) = 1
    ELSE
      LET T(i,j) = -1
    END IF
  NEXT j
NEXT i
END SUB

SUB recombine(s$( ),L2,npop,ntot)
! choose two strings (genotypes) to recombine
LET i = int(npop*rnd) + 1
DO
  LET j = int(npop*rnd) + 1
LOOP until i <> j
LET size = int(rnd*(L2/2)) + 1
LET pos = int(rnd*L2) + 1
LET s1$ = s$(i)
LET s2$ = s$(j)
IF pos + size <= L2 then
  LET s$(ntot+1) = s1$[1:pos-1]&s2$[pos:pos+size]&s1$[pos+size+1:L2]
  LET s$(ntot+2) = s2$[1:pos-1]&s1$[pos:pos+size]&s2$[pos+size+1:L2]
ELSE
  ! apply periodic eboundarey conditions
  LET pbc = pos + size - L2
  LET s$(ntot+1) = s2$[1:pbc] & s1$[pbc+1:pos-1] & s2$[pos:L2]
  LET s$(ntot+2) = s1$[1:pbc] & s2$[pbc+1:pos-1] & s1$[pos:L2]
END IF
LET ntot = ntot + 2
END SUB

SUB mutate(s$( ),L2,npop,ntot)
LET i = int(rnd*npop) + 1
LET pos = int(rnd*L2) + 1
LET c$ = s$(i)[pos:pos]
IF c$ = "1" then
  LET c$ = "0"
ELSE
  LET c$ = "1"
END IF
LET s$(ntot + 1) = s$(i)[1:pos-1] & c$ & s$(i)[pos+1:L2]
LET ntot = ntot + 1
END SUB

```

```

SUB convert(a$,L,s(,))
! converts strings of 0's and 1's to 2D array of spins
! that is, genotype to phenotype
FOR i = 1 to L
  FOR j = 1 to L
    LET n = (j-1)*L + i
    IF a$[n:n] = "1" then
      LET s(i,j) = 1
    ELSE
      LET s(i,j) = -1
    END IF
  NEXT j
NEXT i
END SUB

SUB energy(L,s(,),T(,),E)
LET E = 0
FOR i = 1 to L
  LET ip = i + 1
  IF ip > L then LET ip = 1
  FOR j = 1 to L
    LET jp = j + 1
    IF jp > L then LET jp = 1
    LET n = (j-1)*L + i
    LET E = E - T(n,1)*s(i,j)*s(ip,j) - T(n,2)*s(i,j)*s(i,jp)
  NEXT j
NEXT i
END SUB

SUB selection(s$(,),L,L2,T(,),npop,ntot,Eselect())
DIM s(30,30),save$(1000),Elist(0:1000)
LET Esum = 0
LET Elist(0) = 0
FOR i = 1 to ntot
  CALL convert(s$(i),L,s(,))
  CALL energy(L,s(,),T(,),E)
  LET Esum = Esum - E + 2*L2      ! contribution to Esum > 0
  LET Elist(i) = Esum
NEXT i
MAT save$ = s$
! select new population
FOR ipop = 1 to npop
  LET E = Esum*rand
  LET i = 0
DO

```

```

        LET i = i + 1
    LOOP until E < Elist(i)    ! choose according to energy
    LET s$(ipop) = save$(i)
    LET Eselect(ipop) = Elist(i-1) - Elist(i) + 2*L2
NEXT ipop
END SUB

SUB showoutput(s$(),npop,Eselect())
FOR ipop = 1 to npop
    PRINT ipop,s$(ipop),Eselect(ipop)
NEXT ipop
PRINT
END SUB

```

*Problem 15.18.* Ground state of Ising-like models

- a. Use Program `genetic` to find the ground state of the ferromagnetic Ising model for which  $T_{ij} = 1$  and the ground state energy is  $E = -2L^2$ . Choose  $L = 4$ , and consider a population of 20 strings, with 10 recombinations and 4 mutations per generation. How long does it take to find the ground state energy? You might wish to modify the program slightly so that each new generation is shown on the screen and a pause statement is added so that you can look at the new generations as they appear.
- b. Find the mean number of generations needed to find the ground state for  $L = 4, 6$ , and 8. Repeat each run at least twice. Use a population of 100, a recombination rate of 50% and a mutation rate of 20%. Are there any general trends as  $L$  is increased? How do your results change if you double the population size? What happens if you double the recombination rate or mutation rate? Use larger lattices if you have sufficient computer resources.
- c. Repeat part (b) for the antiferromagnetic model.
- d. Repeat part (b) for the spin glass model where  $T_{ij} = \pm 1$  randomly. In this case we do not know the ground state energy in advance. What criteria can you use to terminate a run?

One of the important features of the genetic algorithm is that the change in the genetic code is selected not in the genotype directly, but in the phenotype. Note that the way we change the strings (particularly with recombination) is not closely related to the two-dimensional lattice of spins. Indeed, we could have used some other prescription for converting our string of 0's and 1's to a configuration of spins on a two-dimensional lattice. If the phenotype is a three-dimensional lattice, we could use the same procedure for modifying the genotype, but a different prescription for converting the genetic sequence (the string of 0's and 1's) to the phenotype (the three-dimensional lattice of spins). The point is that it is not necessary for the genetic coding to mimic the phenotypic expression. This point becomes distorted in the popular press when a gene is tied to a particular trait, because specific pieces of DNA rarely correspond directly to any explicitly expressed trait in the phenotype.

## 15.6 Overview and Projects

All of the models we have discussed in this chapter have been presented in the form of a computer algorithm rather than in terms of a differential equation. These models are an example of the development of a “computer culture” and are a reflection of the way that technology affects the way we think (cf. Vichniac). Can you discuss the models in this chapter without thinking about their computer implementation? Can you imagine understanding these models without the use of computer graphics?

We have given only a brief introduction to cellular automata and other models that are relevant to the newly developing study of complexity, and there are many models and applications that we have not discussed. These models range from attempts to understand the most fundamental processes of nature such as biological evolution and fluid flow to practical studies of the setting of cement (cf. Bentz et al.). In addition, one of the major motivations for the study of cellular automata is their relation to theories of computation and the development of new computer architectures (cf. Hillis). Some researchers believe that ultimately all models of nature can be reduced to cellular automata. One of the attractive features of these models is that the complexity of nature can be ultimately understood as the result of simple and local rules of evolution.

*Project 15.19.* Cellular automata spring-block model of earthquakes

Mechanical models of earthquakes based on Newton’s equations of motion are difficult to simulate because of the wide range of time scales inherent in these models (cf. Carlson and Langer). For this reason various cellular automata models have been proposed (cf. Klein et al.) that approximate the dynamics on short time scales. The idea is that such approximations are not important because the time interval between earthquakes is much larger than the time of an individual slip. Consider a set of  $L$  blocks that are constrained to move in one dimension. Each block is connected to its two nearest neighbors by harmonic springs with stiffness constant  $k_c$  and to a loader plate by a leaf spring with stiffness constant  $k_L$ . The loader plate moves at velocity  $v$ . The stress  $\sigma_j$  on block  $j$  at time  $t$  after the external loader plate has advanced a distance  $nv$  is given by

$$\sigma_j = k_L(nv - x_j(t)) + k_c(x_{j+1}(t) + x_{j-1}(t) - 2x_j(t)), \quad (15.12)$$

where  $x_j(t)$  is the displacement of block  $j$  at time  $t$  from its initial equilibrium position. The time  $t$  is associated with local rupture and stress release, which typically is the order of a few seconds. The integer  $n$  is the number of loader plate updates and is the time associated with the transmission of the tectonic loading force, typically the order of years or decades. We assume that the equilibrium distance between blocks is unity.

A block slips only if its stress exceeds the static friction threshold in which case it advances by an amount proportional to the stress. The displacement of block  $j$  at time  $t + 1$  is given by the equation of motion:

$$x_j(t + 1) = x_j(t) + \frac{\sigma_j(t)}{k_j} \theta(\sigma_j(t) - S), \quad (15.13)$$

where

$$k_j = k_L + q_j k_c, \quad (15.14)$$

and  $S$  is the static friction threshold. The value of the coordination number  $q$  in (15.14) is two except if free boundary conditions are used in which case  $q = 1$  for the end blocks. The step function  $\theta(x)$  is unity for  $x \geq 0$  and zero for  $x < 0$ . Note that the equation of motion sets the residual stress on block  $j$  equal to zero. The evolution of this model is given by the following steps.

1. Choose the initial displacements of the blocks at random, that is, let  $x_j(t = 0) = (r_j - 0.5)$ , where  $r_j$  is a uniform random number in the unit interval. Set  $n = 0$ .
2. Compute the stress  $\sigma_j$  on each block according to (15.12).
3. Blocks  $j$  for which  $\sigma_j \geq S$  slip according to the rule,  $x_j \rightarrow x_j + \sigma_j/k_j$  (see (15.13)). Increment the “slip” time by unity,  $t \rightarrow t + 1$ .
4. Repeat steps 2 and 3 until  $\sigma_j < S$  for all blocks.
5. Increase the stress on each block by the amount  $k_L v \Delta n$  and increase  $n$  by  $\Delta n = 1$ .
6. Repeat steps 2–5 until a sufficient number of loader plate updates have been obtained.

Note that the evolution is deterministic and that randomness is introduced only in the initial conditions. The nonlinearity in the model is in the rule for slips. In the following, we set  $k_L = k_c = 1$ .

- a. Explain how the dynamics of the model can lead to avalanches of failed blocks. Each avalanche of failed blocks represents an earthquake. Write a program to implement the model. For your preliminary simulations, choose a system of  $L = 64$  blocks. Equilibrate the system for a time long enough for each block to fail (slip) at least once. Choose  $S = 1$  and  $v = 0.02$ . After equilibrium has been established, compute  $s$ , the number of blocks that slip during one loader update. (If a block slips more than once, count it each time it slips.) Compute the histogram  $H(s)$  averaged over many loader plate updates. How would you characterize the dependence of  $H(s)$  on  $s$ ? Do you see any evidence of power law behavior? Does the behavior of  $H(s)$  depend on the size of the system? Choose either periodic or free boundary conditions and determine if the boundary conditions influence your results.
- b. Another quantity of interest is the average slip deficit  $\phi_n$  defined as

$$\phi_n = \frac{1}{L} \sum_j (x_j - nv). \quad (15.15)$$

The average slip deficit is a measure of the lag of the blocks with respect to the movement of the loader plate. Describe the nature of the dependence of  $\phi_n$  on  $n$ . Is the  $n$ -dependence of  $\phi_n$  purely random? Compute the power spectrum of  $\phi_n$  and characterize its dependence on  $\omega$ . Try different values of the parameters  $v$ ,  $S$ , and  $k_c$ . The properties of this and similar earthquake models are an area of much current interest.

## References and Suggestions for Further Reading

- Preben Alstrøm and João Leão, “Self-organized criticality in the game of life,” *Phys. Rev. E* **49**, R2507 (1994). The authors find evidence for self-organized criticality using finite size scaling, but suggest that further studies are needed.
- P. Bak, “Catastrophes and self-organized criticality,” *Computers in Physics* **5**(4), 430 (1991). A good introduction to self-organized critical phenomena.
- Per Bak, Kan Chen, and Michael Creutz, “Self-organized criticality in the Game of Life,” *Nature* **342**, 780 (1989). These workers consider lattices up to  $150 \times 150$  and find evidence that the local configurations self-organize into a critical state.
- Per Bak and Kim Sneppen, “Punctuated equilibrium and criticality in a simple model of evolution,” *Phys. Rev. Lett.* **71**, 4083 (1993); Henrik Flyvbjerg, Kim Sneppen, and Per Bak, “Mean field theory for a simple model of evolution,” *Phys. Rev. Lett.* **71**, 4087 (1993);
- P. Bak, C. Tang, and K. Wiesenfeld, “Self-organized criticality,” *Phys. Rev. A* **38**, 364 (1988).
- Per Bak and Michael Creutz, “Fractals and self-organized criticality,” in *Fractals in Science*, Armin Bunde and Shlomo Havlin, editors, Springer-Verlag (1994).
- Charles Bennett and Marc S. Bourzutschky, “Life not critical?,” *Nature* **350**, 468 (1991). The authors present evidence that the criticality of the Game of Life is an artifact resulting from the simulation of small lattices.
- Dale P. Bentz, Peter V. Coveny, Edward J. Garboczi, Michael F. Kleyn, and Paul E. Stutzman, “Cellular automaton simulations of cement hydration and microstructural development,” *Modelling Simul. Mater. Sci. Eng.* **2**, 783 (1994).
- E. R. Berlekamp, J. H. Conway, and R. K. Guy, *Winning Ways for your Mathematical Plays*, Vol. 2, Academic Press (1982). A discussion is given of how the Game of Life simulates a universal computer.
- J. M. Carlson and J. S. Langer, “Mechanical model of an earthquake fault,” *Phys. Rev. A* **40**, 6470 (1989).
- John W. Clark, Johann Rafelski, and Jeffrey V. Winston, “Brain without mind: Computer simulation of neural networks with modifiable neuronal interactions,” *Phys. Repts.* **123**, 215 (1985).
- Michael Creutz, “Deterministic Ising dynamics,” *Ann. Phys.* **167**, 62 (1986). A deterministic cellular automaton rule for the Ising model is introduced.
- Gary D. Doolen, Uriel Frisch, Brosl Hasslacher, Steven Orszag, and Stephen Wolfram, editors, *Lattice Gas Methods for Partial Differential Equations*, Addison-Wesley (1990). A collection of reprints and original articles by many of the leading workers in lattice gas methods.

- Stephanie Forrest, editor, *Emergent Computation: Self-Organizing, Collective, and Cooperative Phenomena in Natural and artificial Computing Networks*, MIT Press (1991).
- Stephen I. Gallant, *Neural Network Learning and Expert Systems*, MIT Press (1993).
- M. Gardner, *Wheels, Life and other Mathematical Amusements*, W. H. Freeman (1983).
- G. Grinstein and C. Jayaprakash, "Simple models of self-organized criticality," *Computers in Physics* **9**, 164 (1995).
- G. Grinstein, Terence Hwa, and Henrik Jeldtoft Jensen, " $1/f^\alpha$  noise in dissipative transport," *Phys. Rev. A* **45**, R559 (1992).
- B. Hayes, "Computer recreations," *Sci. Amer.* **250**(3), 12 (1984). An introduction to cellular automata.
- Jan Hemmingsson, "Consistent results on 'Life'," *Physica D* **80**, 151 (1995). The author measures the same properties as Bak, Chen, and Creutz and finds that the power law behavior seen for smaller lattices disappears for larger lattices ( $1024 \times 1024$  with open boundary conditions).
- John Hertz, Anders Krogh, and Richard G. Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley (1991).
- J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. Natl. Acad. Sci. USA* **79**, 2554 (1982).
- W. Daniel Hillis, *The Connection Machine*, MIT Press (1985). A discussion of a new massively parallel computer architecture influenced in part by physical models of the type discussed in this chapter.
- H. M. Jaeger, Chu-heng Liu, and Sidney R. Nagel, "Relaxation at the angle of repose," *Phys. Rev. Lett.* **62**, 40 (1989). The authors discuss their experiments on real sand piles.
- Stuart A. Kauffman, *The Origins of Order: Self-Organization and Selection in Evolution*, Oxford University Press (1993); "Cambrian explosion and Permian quiescence: Implications of rugged fitness landscapes," *Evol. Ecol.* **3**, 274 (1989).
- W. Klein, C. Ferguson, and J. B. Rundle, "Spinodals and scaling in slider block models," in *Reduction and Predictability of Natural Disasters*, J. B. Rundle, D. L. Turcotte, and W. Klein, editors, Addison-Wesley (1995).
- J. A. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press (1992).
- Chris Langton, "Studying artificial life with cellular automata," *Physica D* **22**, 120 (1986). See also Christopher G. Langton, editor, *Artificial Life*, Addison-Wesley (1989); Christopher G. Langton, Charles Taylor, J. Doyne Farmer, and Steen Rasmussen, editors, *Artificial Life II*, Addison-Wesley (1989); Christopher G. Langton, editor, *Artificial Life III*, Addison-Wesley (1994);

- Roger Lewin, *Complexity: life at the edge of chaos*, MacMillan (1992). A popular exposition on complexity theory.
- Elaine S. Oran and Jay P. Boris, *Numerical Simulation of Reactive Flow*, Elsevier Science Publishing (1987). Although much of this book assumes an understanding of fluid dynamics, there is much discussion of simulation methods and of the numerical solution of the differential equations of fluid flow.
- Sergei Maslov, Maya Paczuski, and Per Bak, "Avalanches and  $1/f$  noise in evolution and growth models," *Phys. Rev. Lett.* **73**, 2162 (1994).
- William Poundstone, *The Recursive Universe*, Contemporary Books (1985). A book on the Game of Life that attempts to draw analogies between the patterns of Life and ideas of information theory and cosmology.
- Daniel H. Rothman and Stéphane Zaleski, "Lattice-gas models of phase separation: interfaces, phase transitions, and multiphase flow," *Rev. Mod. Phys.* **66**, 1417 (1994). A comprehensive review paper.
- David E. Rumelhart and James L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1: *Foundations*, MIT Press (1986). See also Vol. 2 on applications.
- L. Schulman and P. Seiden, "Statistical mechanics of a dynamical system based on Conway's Game of Life," *J. Stat. Phys.* **19**, 293 (1978).
- Dietrich Stauffer, "Cellular Automata," Chapter 9 in *Fractals and Disordered Systems*, Armin Bunde and Shlomo Havlin, editors, Springer-Verlag (1991). Also see "Programming cellular automata," *Computers in Physics* **5**(1), 62 (1991).
- Daniel L. Stein, editor, *Lectures in the Sciences of Complexity*, Vol. 1, Addison-Wesley (1989); Erica Jen, editor, *Lectures in Complex Systems*, Vol. 2, Addison-Wesley (1990); Daniel L. Stein and Lynn Nadel, editors, *Lectures in Complex Systems*, Vol. 3, Addison-Wesley (1991).
- Patrick Sutton and Sheri Boyden, "Genetic algorithms: A general search procedure," *Amer. J. Phys.* **62**, 549 (1994). This readable paper discusses the application of genetic algorithms to Ising models and function optimization.
- Tommaso Toffoli and Norman Margolus, *Cellular Automata Machines – A New Environment for Modeling*, MIT Press (1987). See also Norman Margolus and Tommaso Toffoli, "Cellular Automata Machines," in the volume edited by Doolen et al. (see above).
- D. J. Tritton, *Physical Fluid Dynamics*, second edition, Oxford Science Publications (1988). An excellent introductory text that integrates theory and experiment. Although there is only a brief discussion of numerical work, the text provides the background useful for simulating fluids.
- G erard Y. Vichniac, "Cellular automata models of disorder and organization," in *Disordered Systems and Biological Organization*, E. Bienenstock, F. Fogelman Soulie, and G. Weisbuch, eds. Springer-Verlag (1986). See also G erard Y. Vichniac, "Taking the computer seriously

in teaching science (an introduction to cellular automata),” in *Microscience*, Proceedings of the UNESCO Workshop on Microcomputers in Science Education, G. Marx and P. Szucs, editors, Balaton, Hungary (1985).

M. Mitchell Waldrop, *Complexity: the emerging science at the edge of order and chaos*, Simon and Schuster (1992). A popular exposition of complexity theory.

Stephen Wolfram, editor, *Theory and Applications of Cellular Automata*, World Scientific (1986).

A collection of research papers on cellular automata that range in difficulty from straightforward to specialists only. An extensive annotated bibliography also is given. Two papers in this collection that discuss the classification of one-dimensional cellular automata are S. Wolfram, “Statistical mechanics of cellular automata,” *Rev. Mod. Phys.* **55**, 601 (1983), and S. Wolfram, “Universality and complexity in cellular automata,” *Physica* **B10**, 1 (1984).

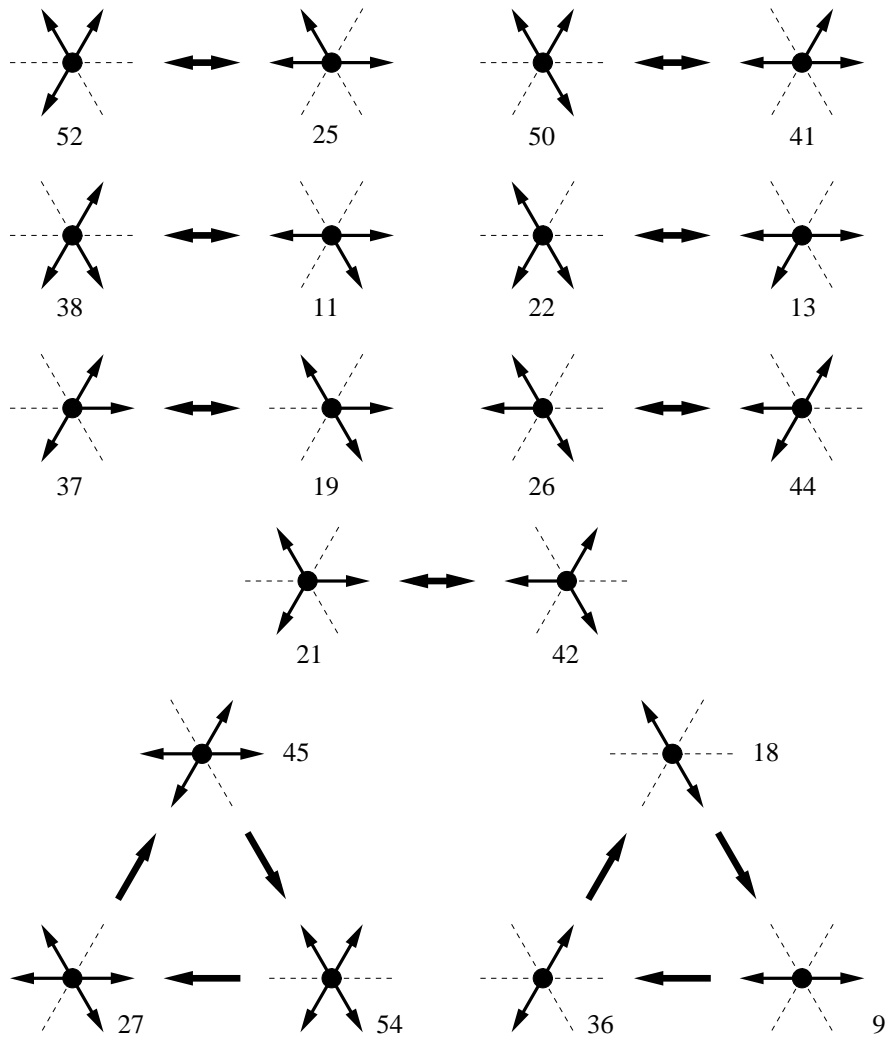


Figure 15.4: Examples of collision rules for a lattice gas on a triangular lattice. The rule for configurations that are not shown is that the velocities do not change after a collision. The numbers represent the way that the velocities at a lattice site are encoded.