The Ising model cellular automata is Problem 15.5, but you may need to read previous pages

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 then 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 $z = 1$.

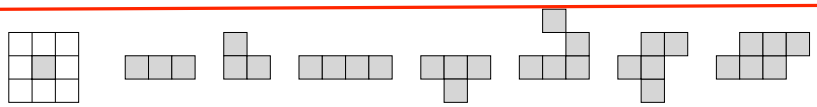

Figure 15.2: (a) The local neighborhood of a site is given by the sum of its eight neighbors. (b) Examples of initial configurations for the Game of Life, some of which lead to interesting patterns. Live cells are shaded.

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"
```

```
               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

a. `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?

b. 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?

c. 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 ith box. The quantity S is given by

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

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

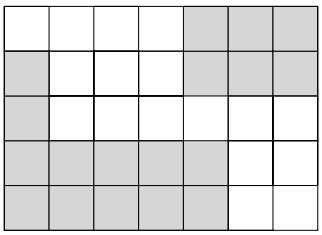

Figure 15.3: What is the evolution of these clusters using the rule discussed in Problem 15.4a. The shaded squares correspond to the 1-sites.

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 ± 1 instead of 0 and 1. The energy is given by

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

The notation $j = \mathrm{nn}(i)$ 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

$$m = \frac{1}{N} \sum_{i=1}^{N} s_i. \tag{15.3}$$

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, $s_i \to -s_i$, 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 40×40 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.