CS 420/594: Complex Systems & Self-Organization
Project 1: "Edge of Chaos" in 1D Cellular Automata

Due: Sept. 15

Introduction

In this project you will explore "Edge of Chaos" phenomena (Wolfram class IV behavior) in 1D cellular automata. You will do this by systematically modifying randomly generated transition tables and observing the lambda and entropy values associated with phase changes in the behavior of the automata.

Experimental Setup

Simulators

You will be using the **ca** simulator provided with Flake's *Computational Beauty of Nature*. There are several different versions that you can use; it's up to you.

- **Java version**. You can execute this directly from the CBN website <http://mitpress.mit.edu/books/FLAOH/cbnhtml/java.html>. There is also a local copy of the jar file (~mclennan/pub/420-594/CBN/cbn-java.jar or <http://www.cs.utk.edu/~mclennan/Classes/420/experiments/CBN/cbn-java.jar>), which you can copy to your own computer to use offline.
- **Unix version**. Runs on SunOS. You can get it from the CBN website <http://mitpress.mit.edu/books/FLAOH/cbnhtml/download.html> or locally (`<~mclennan/pub/420-594/CBN/cbn/code/bin/ca>`, <http://www.cs.utk.edu/~mclennan/Classes/420/experiments/CBN/cbn/code/bin>). You may want to copy the **ca** program into your own directory.
- **Mac version**. Runs under Mac OS 9/X. You can get it from the CBN website <http://mitpress.mit.edu/books/FLAOH/cbnhtml/download.html>.
- **Windows version**. You can get it from the CBN website <http://mitpress.mit.edu/books/FLAOH/cbnhtml/download.html>.

As I said, it's up to you which you use. The Java version has a nice graphical interface; the others have a command-line interface, as described in the book (p. 256, but note some changes as described in the Errata). You will probably be doing some cutting and pasting of rule strings between the simulator and your □-H calculator (see below), so you will have to find a way of doing the experiments that is not too tedious. You might want to write a shell script to automate some of the procedure. (Note that the command-line versions have a `-help` option.)

Table-walk-through Procedure

As explained in class, we cannot simply allow the simulator to pick a random rule string with a certain □ value, since there is too much variability between unrelated rule strings. Therefore we will use Chris Langton's table-walk-through method, which involves generating a series of rule strings differing only in the number of quiescent entries. The following describes an operational procedure.

Pick a random seed and record it. See the last page of this handout for a form that you can use for recording your experiments; for your convenience, a blank form is available online:
<http://www.cs.utk.edu/~mclennan/Classes/420/handouts/Experiment-Record-1.doc>.

Each experiment is conducted on a random rule string. The easiest way to get this is to set `lambda`=1.0 in the **ca** simulator, and let it generate a random rule string. (You can also generate one yourself by picking 13 random integers in the range 1 to 4; we exclude the quiescent state.) If you have **ca** generate the string, you may have to replace one or more zeroes by digits in the range 1–4. Make sure to record your random rule string.

You will now *decimate* your rule string by zeroing one entry at a time. For your original rule string and for each of the decimated rule strings you will compute a □ and an H value. Whether you compute them as you go along or do them all at then end is up to you; which is easier will probably depend on the cutting-and-pasting or other experimental procedure that you have selected. You will also observe and record the behavior (I, II, III, or IV) of the original string and each of its decimations.

For each decimation, randomly select one of the non-zero entries in the rule string and set it to zero. Observe the behavior of the resulting CA and record it. Note: if you are using the Java version of the simulator, make sure you set `lambda=-1` (or any negative value) before you enter your decimated string, or it will ignore your string and regenerate the undecimated string!

□ and Entropy Calculations

The □ and entropy values are defined over the complete transition table, which has $T = K^N$ (for $N = 2r + 1$) entries. However, we have an abbreviated rule-string, of length $(K \square 1)N + 1$, which gives the new state R_k for the sum k of the neighborhood states; in most cases there are multiple configurations having the same sum.

Therefore, let C_k be the number of neighborhood configurations having the sum k. For K=5 and r=1, C_k is given by the following table:

k	0	1	2	3	4	5	6	7	8	9	10	11	12
C_k	1	3	6	10	15	18	19	18	15	10	6	3	1

Define n_s to be the number of configurations leading to state s:

$$n_s = \square_{\{k \mid R_k = s\}} C_k ,$$

where R_k is the kth character (counting from 0) of the rule string. That is, n_s is the total number of configurations that are mapped into state s by the rule string. The probability of a new state in the complete transition table is given by $p_s = n_s/T$. Then Langton's λ is defined:

$$\lambda = \frac{T - n_0}{T} = 1 - {}^{n_0}\!/_{T} = 1 - p_0.$$

The entropy of the complete transition table is defined:

$$H = -\sum_s p_s \lg p_s,$$

where $\lg x$ is the logarithm of x to the base 2. (Note that $0 \lg 0 = 0$, which you will have to handle as a special case since $\lg 0 = -\infty$.)

Since you will be computing the λ and H values for each rule string, you will have to use a program to do it. This is not an important part of the project, so you can do it in any way convenient. You can implement your own program, or several (or all) of you can get together and share a program. Also, some versions of the ca program will compute λ (but not H) for you. If you know LISP, you can use my program: <`~mclennan/pub/420-594/entropy.lsp`> or <http://www.cs.utk.edu/~mclennan/Classes/420/experiments/entropy.lsp>.

Writeup

Calculations

Compute the average and standard deviation of the λ and entropy values for all simulation instances that exhibit class IV behavior. Which (λ or H) seems to be a more reliable indicator of class IV behavior?

Graphs

Make two graphs, one of behavior vs. λ, the other of behavior vs. H. That is, use λ for the abscissa (x-axis) of one graph and H for the abscissa of the other. For the ordinate (y-axis) of both graphs, use the following numerical values to indicate qualitative behavior: 0 for classes I and II, 1 for class IV, and 2 for class III. Each of your graphs should show all of your experiments as separate curves; try to use colors or other ways of making the curves distinguishable.

Discussion

Draw some conclusions about the range of values of λ and H that lead to calls IV behavior. Note any anomalies. Did you ever observe class I or II behavior at high λ and H values? Did you ever observe complex (IV) or chaotic (III) behavior at λ or $\square$ values that were otherwise in the simple (I, II) region? How do you explain these anomalies?

Experiment Record: 1D CA “Edge of Chaos”

Your Name: __Bruce MacLennan__

simulator: ☐ java, ☐ unix, [✓] mac, ☐ windows, ☐ other: ______________
K (states) = 5. *r* (radius) = 1
initial state (init) = __–1___
wrap: [✓] sq: ☐

random seed = __100__. Experiment Number: __1__

Random Rule String:

3	2	3	2	4	1	1	2	3	4	1	1	2
0	1	2	3	4	5	6	7	8	9	10	11	12

Table Walk-through:

Step	**Entry Zeroed**	**Class**	λ	*H*	**Observations**
0	—	III	1.000	1.940	
1	3	III	0.920	2.169	
2	8	III	0.800	2.133	
3	9	III	0.720	2.086	
4	6	$\boxed{\textbf{IV}}$	$\boxed{0.568}$	$\boxed{2.042}$	*looks like about to become periodic*
5	1	II	0.544	2.007	
6	2	II	0.496	1.812	
7	12	II	0.488	1.797	
8	4	II	0.368	1.354	
9	10	II	0.320	1.269	
10	0	I	0.312	1.206	*dies very quickly from here on down*
11	11	I	0.288	1.154	
12	7	I	0.144	0.595	
13	5	I	0.000	0.000	