

Complex Systems I

Cellular Automata

Outline

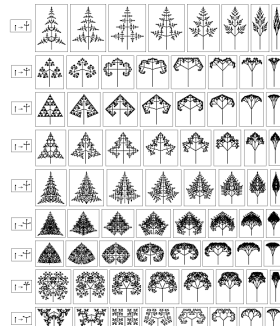
- Complex Systems
 - The science of emergence
- Cellular automata in 1D
- The Game of Life

References for this lecture

- Wolfram, S. (2002). *A New Kind of Science*, Wolfram Media Inc. Champaign, Ill.
 - Wolfram starts with a pictorial introduction to cellular automata and the idea that there are four classes of behaviour produced by simple rule systems: simple, nested, random and "complex".
 - The basic definitions of the 1-dimensional cellular automata and the four classes of behaviour they produce
 - pages 23-41 (from chapter 2)
 - The full range of behaviours for all 256 simple cellular automata
 - pages 51-60 (from chapter 3)
 - Sensitivity of cellular automata to initial conditions
 - pages 223-230 (from chapter 5)
 - How information is transmitted through the different classes of cellular automata
 - 250-254 (from chapter 5)

Emergence

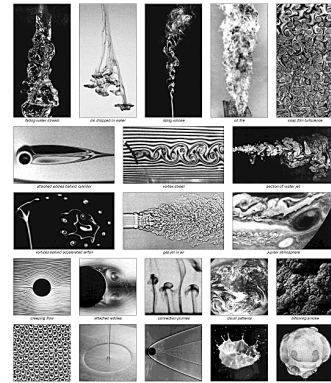
- Simple rules give rise to complex designs



Limiting patterns produced by substitution systems of the type shown in the previous picture. The patterns on each row are obtained from rows that are itself of the same generation with periodicities lengths. The images between the brackets are taken to illustrate by 1/2 in successive pictures across the row. Note that pictures shown in different rows are related differently, so that the overall pattern does not depend on the image height. The similarity between pictures on this page and overall branching patterns and shapes of leaves in many kinds of plants is striking.

Examples from nature

- Many physical systems show complex emergent behaviours



Examples of typical patterns generated in various kinds of fluid flow. Note the frequent occurrence of seemingly random behaviour.

Reductionist vs Emergent Science

- Reductionism breaks systems into their component parts. It looks for regularities in the complexity. It was generally assumed that simple arrangements give rise to simple behaviours, and complex arrangements give rise to complex behaviours.
- Emergent science builds patterns out of components parts without using a central controller. Different types of patterns are observed:
 - Simple patterns do give rise to simple behaviours but they also give rise to complex ones
 - Complex patterns do give rise to complex behaviours, but in some cases the complexity collapses and their support stable simple behaviours.

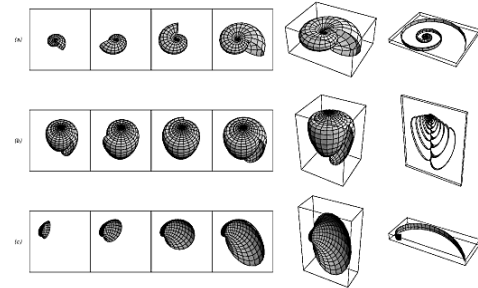
Generative mechanisms

- Emergent science (as opposed to reductionist science) is based on generative mechanisms.
 - That is, simple rules can be viewed as simple programs, and the phenomena they generate can be viewed as computations.
- In linguistics and computer science, the grammar of a language has long been distinguished from the set of all possible sentences that can be generated.
 - The grammar is finite, the language is infinite. In computer science, the most interesting example of generative power is universal computation, in which a small number of states and an infinite memory can perform any possible computation.
- Among theoretical computer scientists, it has long been known that universal computation can be achieved with very simple rules but nobody really made a fuss about it.
 - In logic tables and electrical circuits, the operator NAND is sufficient to duplicate the behaviours of all other operators. It was just a fact we knew.

Emergence cont.

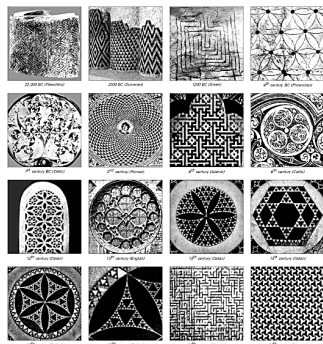
- Emergent science asks questions like:
 - Where does the complexity come from?
 - How easy is it to get?
 - What are the key differences between simple systems and complex systems?

Biological systems also show emergence



Patterns in art

- Repetitive designs can give rise to very complex patterns



Historical examples of non-representational, fractal-like patterns are common and some modern patterns are too. But the more developed kinds of patterns discussed in this chapter do not ever appear to have been used. Note that the second-to-last picture is not an abstract design, but a real-world text written in a highly stylized form of Arabic script.

What sorts of models are useful for thinking about emergence?

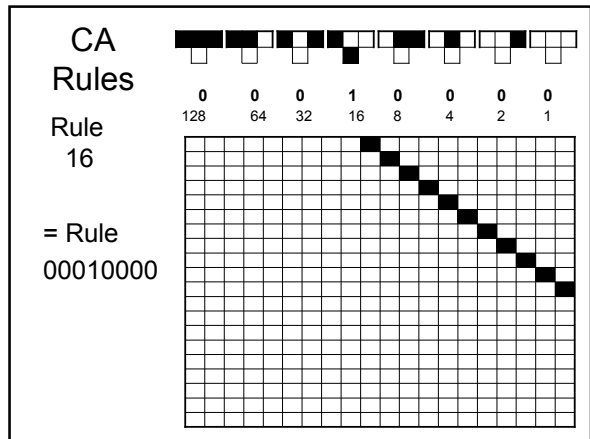
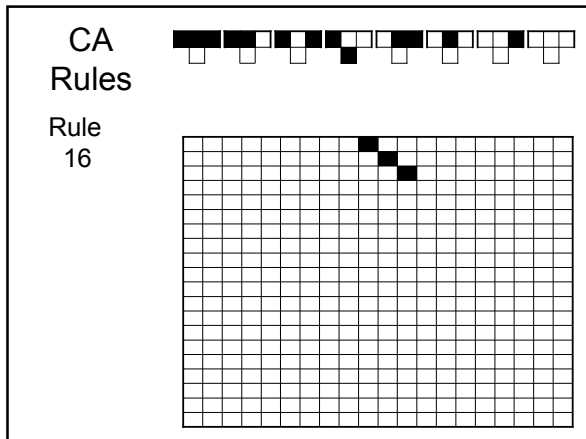
- Neural networks
- Evolutionary computation
- Cellular automata

Cellular Automata

- Simple rules and simple initial conditions can give rise to the most computationally complex behaviour.
- Insights can be gained by studying the space of behaviours of very simple systems.

Cellular automata definition

- Imagine a page of graph paper in which each small square can be coloured black or white.
- Cellular automata are simple rules that tell a square what colour it should be.
- The simplest cellular automata start with a single black square at the top centre of the page.
 - Each row moving down the page looks at the colours of the row above and uses the same rule set to decide its colours.
 - If each square considers just three squares in the previous row - the square above and its right and left neighbours, there are 256 possible rule sets.
 - Each one is given a number from 0 to 255, which when written in binary specifies exactly the rule set

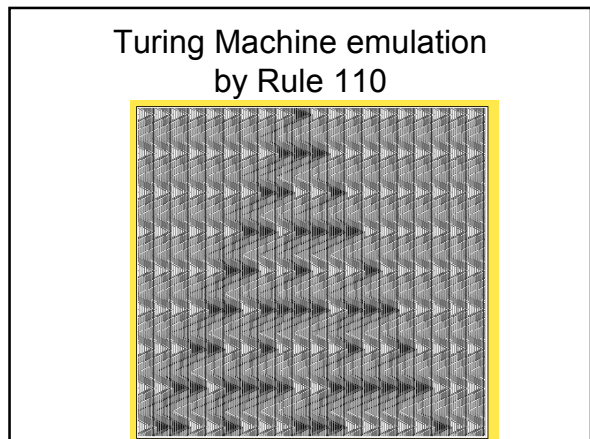
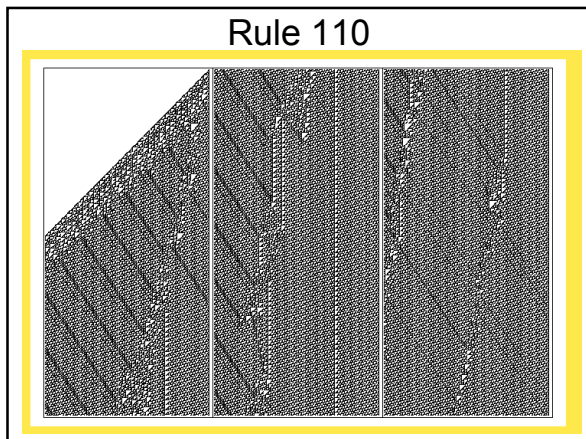


What behaviours emerge?

- From the rules themselves, it is difficult to envisage what patterns they will generate when iterated down the pages.
- Wolfram explored the patterns by trying them out. He discovered that the patterns generated fall into a small number of classes.
 - The simplest are uniform patterns (e.g., rule 0 turns all squares white and rule 255 turns them all black) and repetitive patterns (e.g., rule 50 generates checkerboards).
 - More interesting are nested patterns (e.g., rule 90 generates intricate patterns of nested triangles).
 - However patterns are not restricted to repetitive designs. Some cellular automata produce essentially random patterns (e.g., rule 30 has no overall regularity, and in the first million steps the pattern of squares below the initial black square never repeats itself, and in fact passes statistical tests for randomness).
 - The most interesting patterns are neither completely regular nor random (e.g., rule 110 takes 2780 steps to settle into a periodic pattern 240 steps long, and shows areas of local order interacting with more random areas, forming seemingly regular structures on a larger scale).
- The take home message of the classes is that simple rules generate behaviour ranging from the simple to the complex: the four classes shown include repetition, nesting, randomness and localised structures.
- Traditional science has focussed on analysing simple behaviours against a sea of complexity. Wolfram changes figure and ground to seeing complex behaviours as part and parcel of the space of typical programs.

Demo: NetLogo 1D Cellular Automata

- Tutorial: Cellular automata



Sensitivity to initial conditions: Differences in handling information

The four classes of CAs have characteristic properties that they share, such as similar sensitivities to initial conditions

1. Information about initial conditions is forgotten:
 - The effects of the changes die out and exactly the same changes are reached (eg Rule 160)
2. Information about initial conditions is retained to the final configuration, but is not communicated to other parts of the system
 - The effects of the changes may persist but are localised to a small region (egg rule 108)
3. Long range communication:
 - The effects spread at a uniform rate, eventually affecting every part of the system
4. Intermediate between localised and long range:
 - In class 4 changes can also spread but only in a sporadic way. In principle information can be communicated but only does so if it affects one of the localised structures that moves across the system.

What sorts of results can be established?

- simple programs can produce universal computation
- simple rule systems are ubiquitous, hence universal computation is ubiquitous
- Take home msgs
 - The take home message of Wolfram's NKS is that among the space of programs, simple programs that produce complex behaviours are not rare.
 - The same basic forms of behaviour occur in many different contexts in both real and artificial worlds, leading to the possibility of universal principles to determine overall behaviour.

The Game of Life

- The game board is a rectangular cell array, with each cell either empty or filled.
- At each tick of the clock, the next generation is created by the following rules:
 - if a cell is empty,
 - fill it if 3 of its neighbors are filled (otherwise leave it empty)
 - if a cell is filled, it
 - dies of loneliness if it has 1 or fewer neighbors
 - continues to live if it has 2 or 3 neighbors
 - dies of overcrowding if it has more than 3 neighbors
- Neighbors include the cells on the diagonals.

Demo: NetLogo Life (2D cellular automata)

Take home messages

- Some robust behaviours emerge that are simple, but emerge from complex underlying rules
 - Eg. Random initial starting positions almost always produce gliders.
- Behaviours emerge at a more complex level that are inherently unpredictable at the lower level except by simulating the full model:
 - Eg. whether a particular starting pattern will stop (formally analogous to the Halting problem with Turing machines)

Tutorial

- Work through the tutorials #2 and #3 to get experience with Net Logo commands, or just read the cogs 2010 tutorial if you just want an introduction to CAs.

Command Centre

Read the NetLogo tutorials #2 and #3 to learn about simple programs

- A model has 3 modes for commands:
 - Patches
 - Turtles
 - Observers
- Use the command centre to try:
P> set pcolor pxcor + pycor

Sample commands: Playing with colours

Patch mode

```
P> set pcolor pxcor + pycor
;; sets the colour of a patch to the sum of the x and y coordinates.
;; Colours are specified by numbers between 0 and 140
;; this is a comment !
```

Observer and Turtle modes:

```
O> clear-patches ;; reset the display
O> create-turtles 36 ;; turtles start out at the origin, facing in different directions
T> forward who / 5 ;; who is the number of the turtle from 0 to 35.
;; Move who/5 steps in the direction the turtle is heading
```

Try some other patterns:

```
P> set pcolor 1000 / ( abs ( pxcor * pycor ) + 1)
P> set pcolor ( pxcor * pycor ) * 0.1
P> set pcolor ( 25 / (1 + exp (0 - pxcor * .2 )) - pycor ) ;; this is the sigmoid
equation
```