Cellular Automata

Cellular automata (CA) models epitomize the idea that simple rules can generate complex patterns.

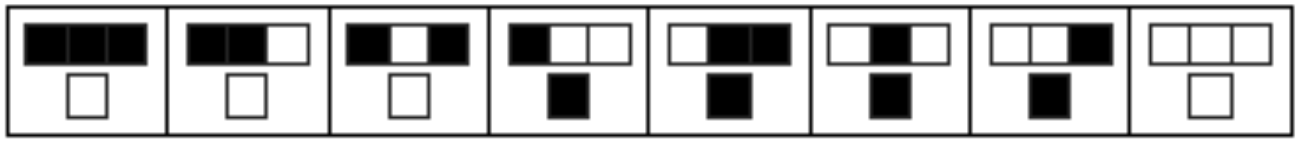
A CA consists of an array of cells each with an integer 'state'.

On each time step a local update rule is applied to the cells. The update rule defines how the a particular cell will update its state as a funcion of its neighbours state.

The CA is run over time and the evolution of the state is observed.

Elementary cellular automata

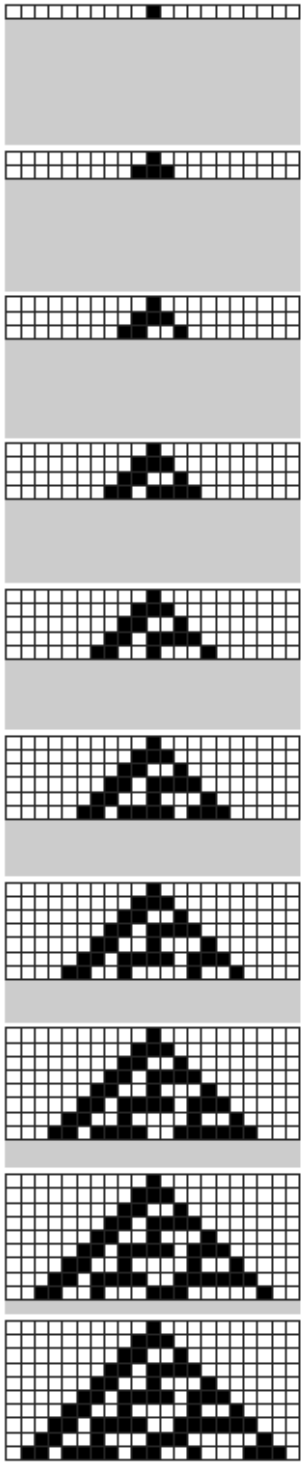
The elementary one dimensional CA is defined by how three cells influence a single cell. For example,



The rules can be expressed in binary form where a 0 represents that a particular configuration gives a white output and a 1 denotes a black output.

The above rule in binary is 00011110. Converting this binary number to base-10, we call this rule 30.

We thus have a set of different rules for elementary cellular automata from 0 up to 255.



Different rules

Elementary CA simulators are easily found on the internet.

For example, a NetLogo implementation can be found at

http://ccl.northwestern.edu/netlogo/

We will look at rules 254, 250, 150, 90, 110, 30 and 193 in this simulator.

Classifying elementary CA

An empirical observation is that cellular automata can be classied according to the complexity and information produced by the behavior of the pattern they produce:

Class 1 : Fixed; all cells converge to a constant black or white set

Class 2 : Periodic; repeats the same pattern, like a loop

Class 3 : Chaotic; pseudo-random

Class 4 : Complex local structures; exhibits behaviors of both class 2 and class 3; with long lived hard to classify structure.

This classication has a bit of a feeling of saying there are three types we roughly understand plus one (class 4) we don't. In particular, classes 1 to 3 can be charcterised by Entropy and the Lyapunov exponent, this is less true of class 4.

Game of Life

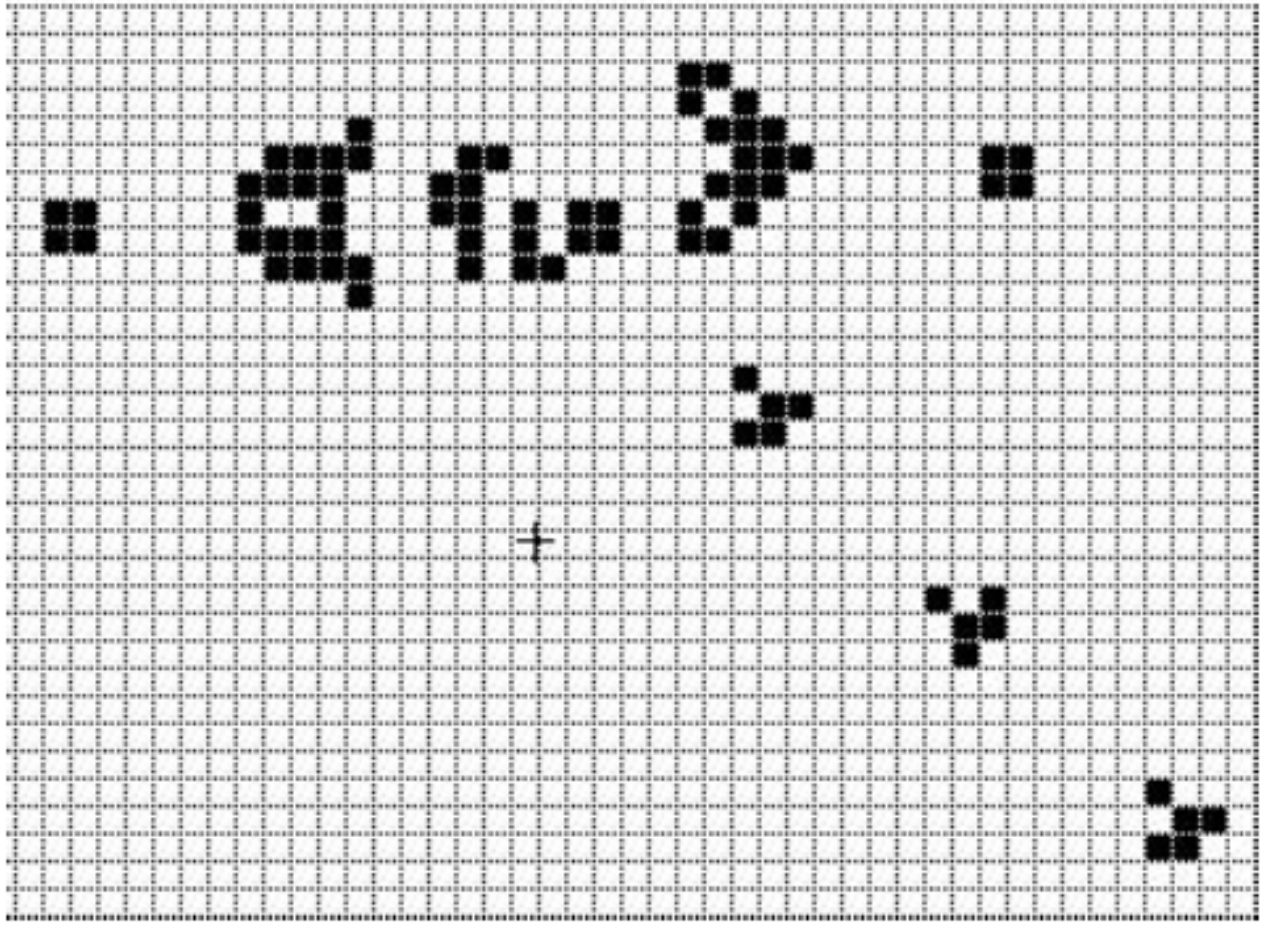
Cellular automata can be extended to have longer range interactions, e.g. the ve nearest neighbours determine the new state of the cell. They can also be extended to two or more dimensions.

A particularly well studied 2D cellular automaton is called The Game of Life. In this CA, each cell checks the state of itself and its eight surrounding neighbors and then sets itself to either alive or dead (hence the name 'Game of Life'). If there are less than two alive neighbors, then the cell dies. If there are more than three alive neighbors, the cell dies. If there are 2 alive neighbors, the cell remains in the state it is in. If there are exactly three alive neighbors, the cell becomes alive. This is process is continued indenitely.

Game of Life is also available at the NetLogo site.

Stable shapes, gliders etc…

There are many interesting patterns generated by this simple rule. One of these is the glider. This set up travels in the same direction as long as it doesn't crash with anything else. Another pictured below is a glider gun, which creates a line of gliders.



Measuring complexity

- The patterns arising from cellular automata give us a lot to think about.
- We see a very simple set of rules generate very complex patterns. Moreover, many of these patterns are life-like. The problem is dening what we mean by life-like and quantifying this idea.
- Doing this is by no means a completed scientic activity.

Kolmogorov Complexity

One of the first definitions of complexity of a string of 1's and 0's was proposed by Kolmogorov in the 1960s. This string may, for example, be the output of a cellular automata.

The basic idea is that the length of the string's shortest description in terms of a computer program is its complexity.

For example, an array of 0s (i.e. 0000000::::) has low complexity, because we can write a program that says 'keep writing 0s'. All periodic arrays have low complexity, since we can write a program that says, for example, "write 0110 ten thousand times".

Kolmogorov Complexity

By Kolmogorov's denition, the strings which are most complex are those which are most difficult to compact without losing information. We denote Kolmogorov complexity with *K.*

This returns us to our second example of entropy. Imagine we can divide our original string in to a finite set of components and label these components A, B, C etc. (example on the board).

We can now replace A, B, C etc. with binary strings 1,01,001,0001, … with the most common strings receiving the shortest encoding.

S is the set of all possible letters and p_s be the probability that letter *s* appears then (by the argument in example 2 of entropy) the length of this coding with be at most its Shannon Entropy plus 1.

$$H = -\sum_{s \in S} p_s \log_2(p_s) + 1$$

Kolmogorov Complexity

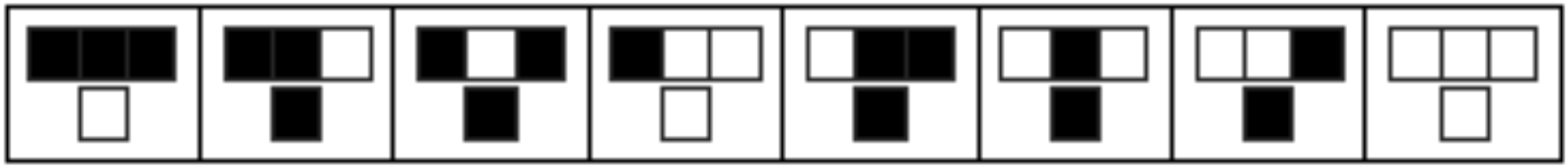
- Random strings then have the highest complexity!
- This goes against our intution of what we mean by complexity.
- There are however examples of strings with high Entropy but low Kolmogorov complexity (for example middle line of rule 30).
- Maybe *H*-*K* is a good measure of complexity?
- But then finding *K* for a given string is difficult!

Universal Computing

- One possible solution to defining complexity in cellular automata is by saying that a CA rule is complex if it it can act as a universal computing machine.
- A universal computing machine (or universal Turing machine) is a computer program that can simulate any other arbitrary computer program to produce the input that program would produce.
- The main point comes down to whether one can construct logical gates with a particular CA rule.

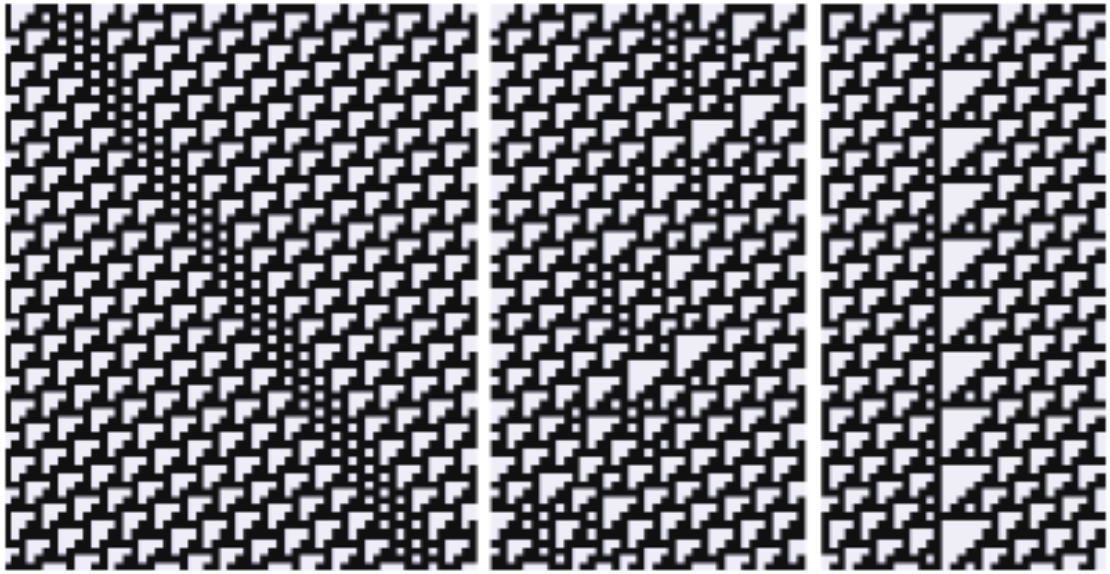
Rule 110

- It turns out that both the 'Game of Life' and even rule 110 from an elementary CA fullfil the criteria for universal computation.
- Rule 110 is,



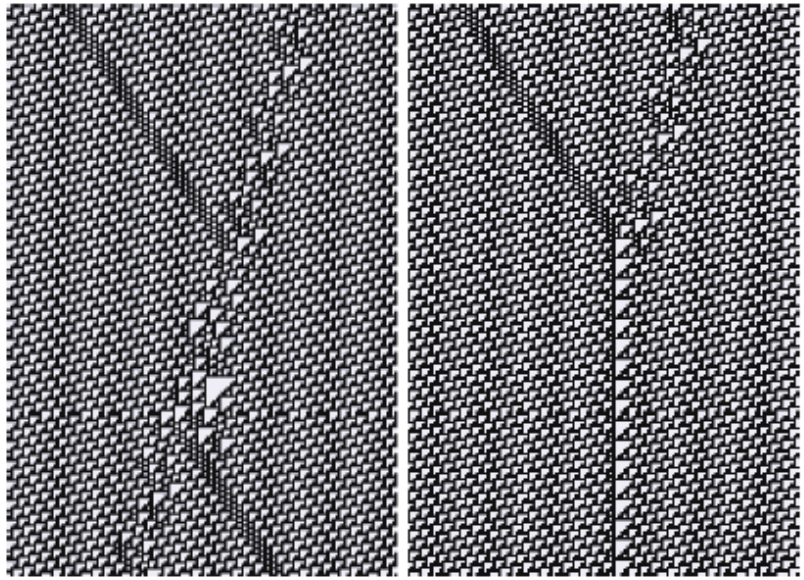
Rule 110

The pictures below give some feeling for how structures in rule 110 can interact to build a computing device. They show a right moving, a left moving,and a stationary strucuture



Rule 110

When interacting, depending on how they meet, they can either pass through each other or cause the formation of the stationary shape.



Putting these sorts of shapes together allow a computer program to be executed, although the complexity of the program itself is likely to be enormous.

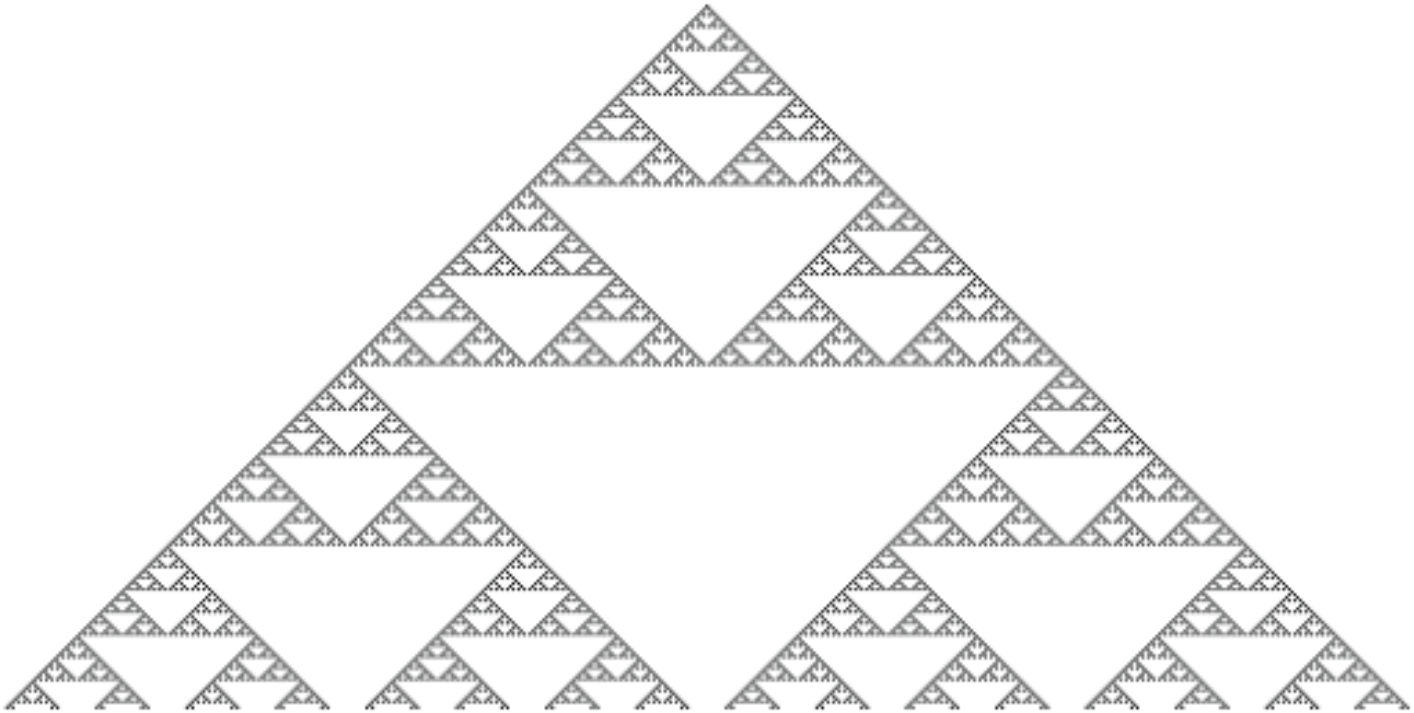
Universal Computing

- Although interesting, this classication of CA in terms of how much computation they can do has not really shed much light on the question of how to denfine and measure complexity.

Box counting dimension

A property of the patterns made by "complex" CA is that they appear to repeat on very dierent scales.

For example, rule 90 gives



Box counting dimension

This structure is identical to a geometric shape known as Sierpinski triangle, which can also be constructed by the following iteration.

The Sierpinksi triangle is one example of a fractal: "a rough or fragmented geometric shape that can be split into parts, each of which is (at least approximately) a reduced-size copy of the whole," (Barnsley, Michael F., Fractals Everywhere 1993).

Box counting dimension

An important property of the Sierpinksi triangle and fractals in general is self-similarity. As we zoom in on the Sierpinski triangle we see another identical triangle and so on forever.

For the Sierpinski triangle constructed using the iterative method, this is a consequence of the way we constructed it. For the cellular automata construction we did not directly encode the self-similarity in the resulting output, but we can now use that self-similarity to analyse the structure.

The main tool for categorising fractals is their box-counting dimension. The basic idea is this. We cover an object with square boxes of some particular size and we ask what is the number of boxes needed to cover all the black parts of the object.

Box counting dimension

Mathematically, let A be a shape or the output array from a CA, $N_\epsilon(A)$ be the number of boxes of side length ϵ needed to cover the shape. If there is a number d so that

$$N_\epsilon(A) \sim \frac{1}{\epsilon^d}$$

as $\epsilon \to 0$, we say that the box-counting dimension of A is d. For d to exist we require that there is a positive constant k such that

$$\lim_{\epsilon \to 0} \frac{N_\epsilon(A)}{1/\epsilon^d} = k$$

Taking the logarithm of both sides and solving for d gives

$$d = -\lim_{\epsilon \to 0} \frac{\ln N_\epsilon(A)}{\ln \epsilon}$$

This gives us a practical means of calculating the box counting dimension. If we calculate $N_\epsilon(A)$ for successively smaller boxes ϵ then the slope of the line in a plot of ϵ vs $\ln N_\epsilon(A)$ gives the box counting dimension.

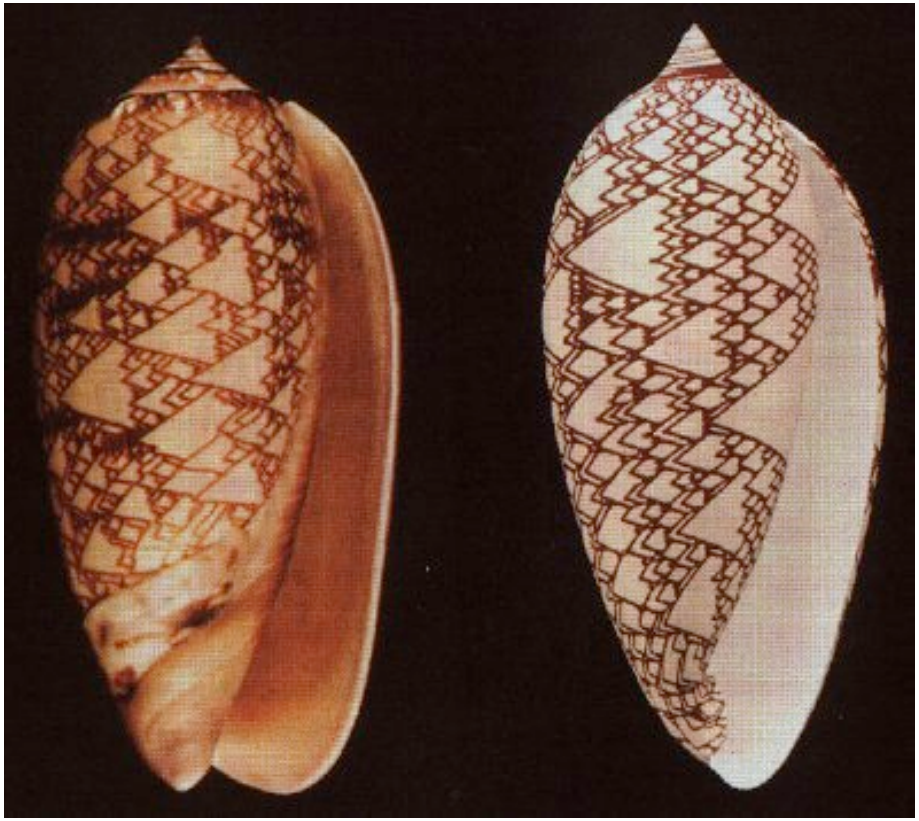
Box counting dimension

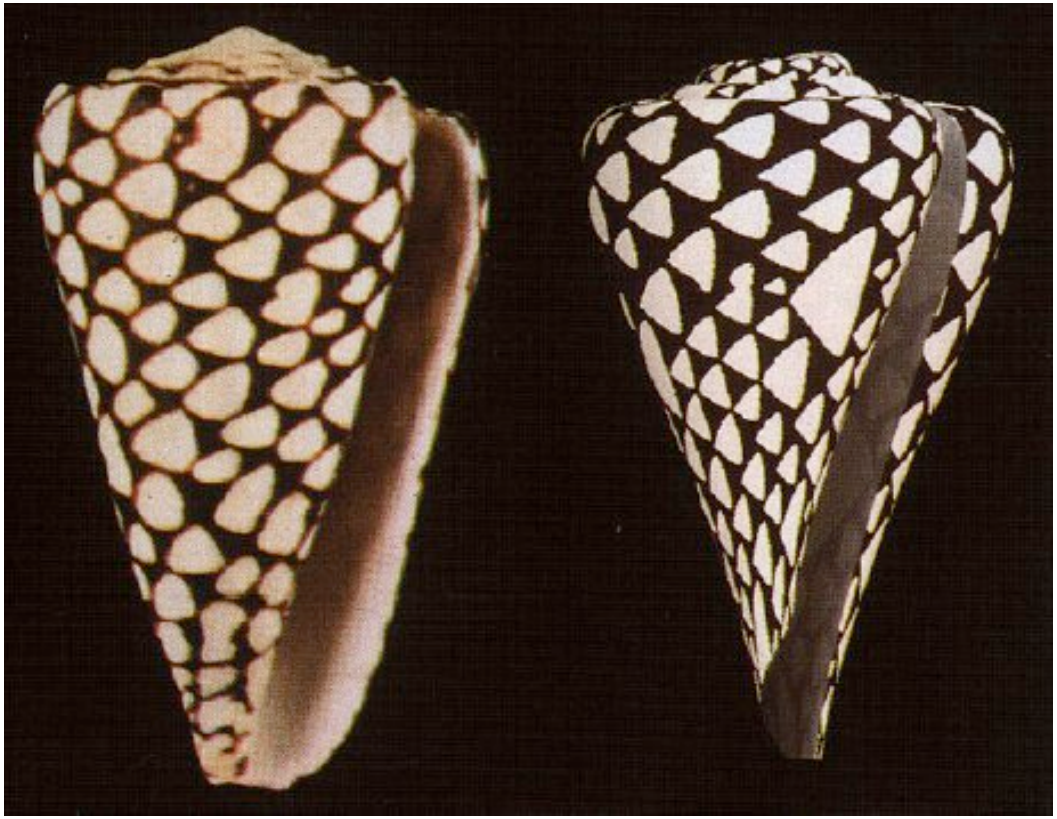
Box counting dimension

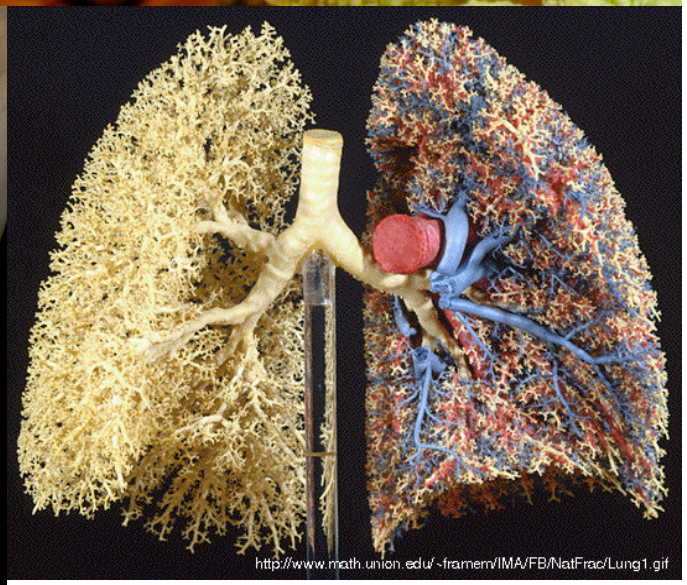
The box counting dimension of a matrix of 0's and 1's can be calculated by repeatedly covering the matrix with a square grid with decreasing lengths l between the joins in the grid. Let N_l be the number of squares in the grid that contains at least one 1. The box counting dimension is

$$d = -\lim_{l \to 0} \frac{\ln(N_l)}{\ln(l)}$$

In practice d can be calculated by plotting $\ln(l)$ vs $\ln(N_l)$ and using least squares regression to find the slope of the straight line relationship between them. Note that although we usually take the limit very small i.e. $l = 1$ or $l = 2$ tends to give a poor estimate of the box counting dimension since the measured structure starts to be on the same scale as the measuring device.

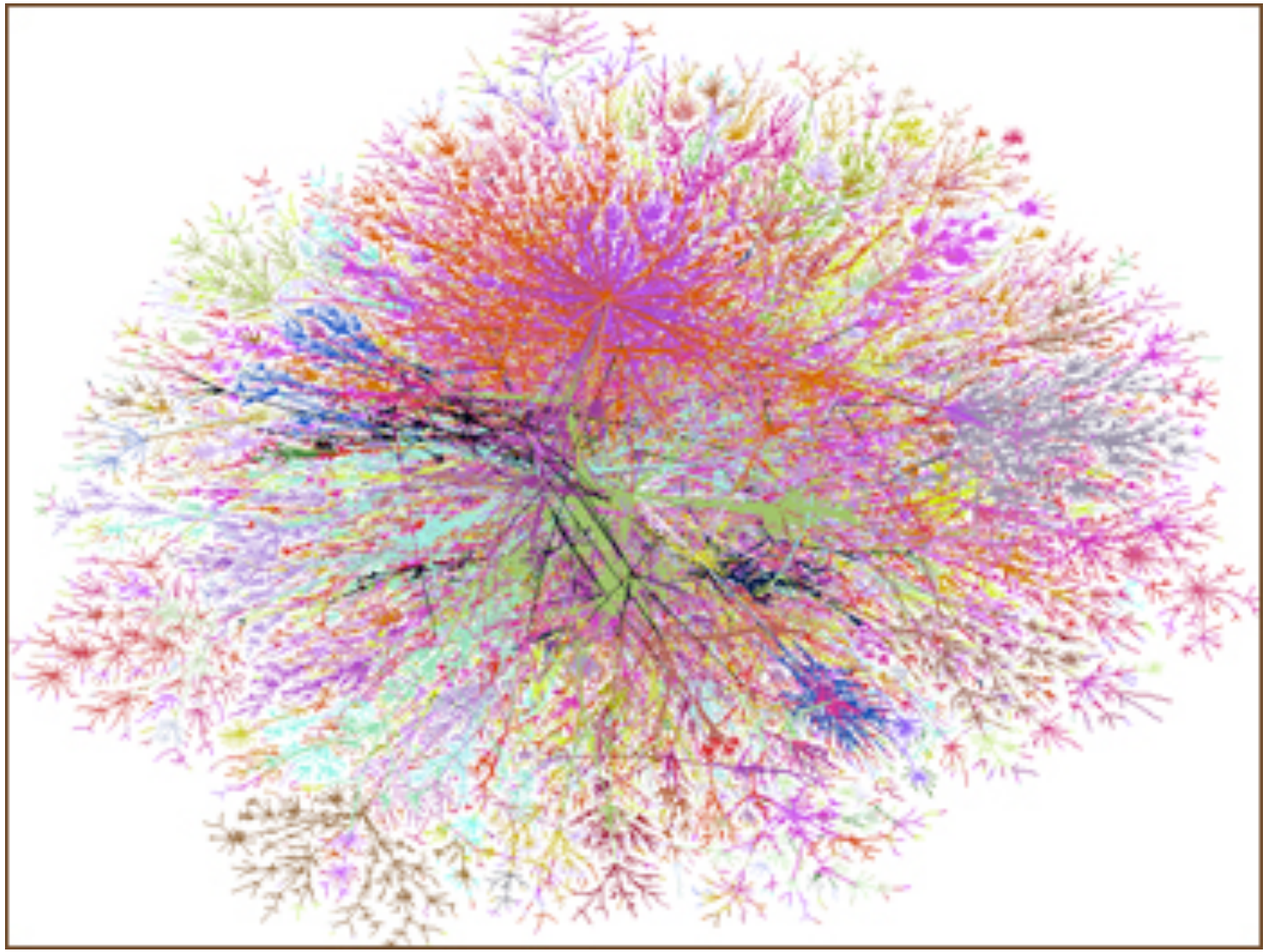


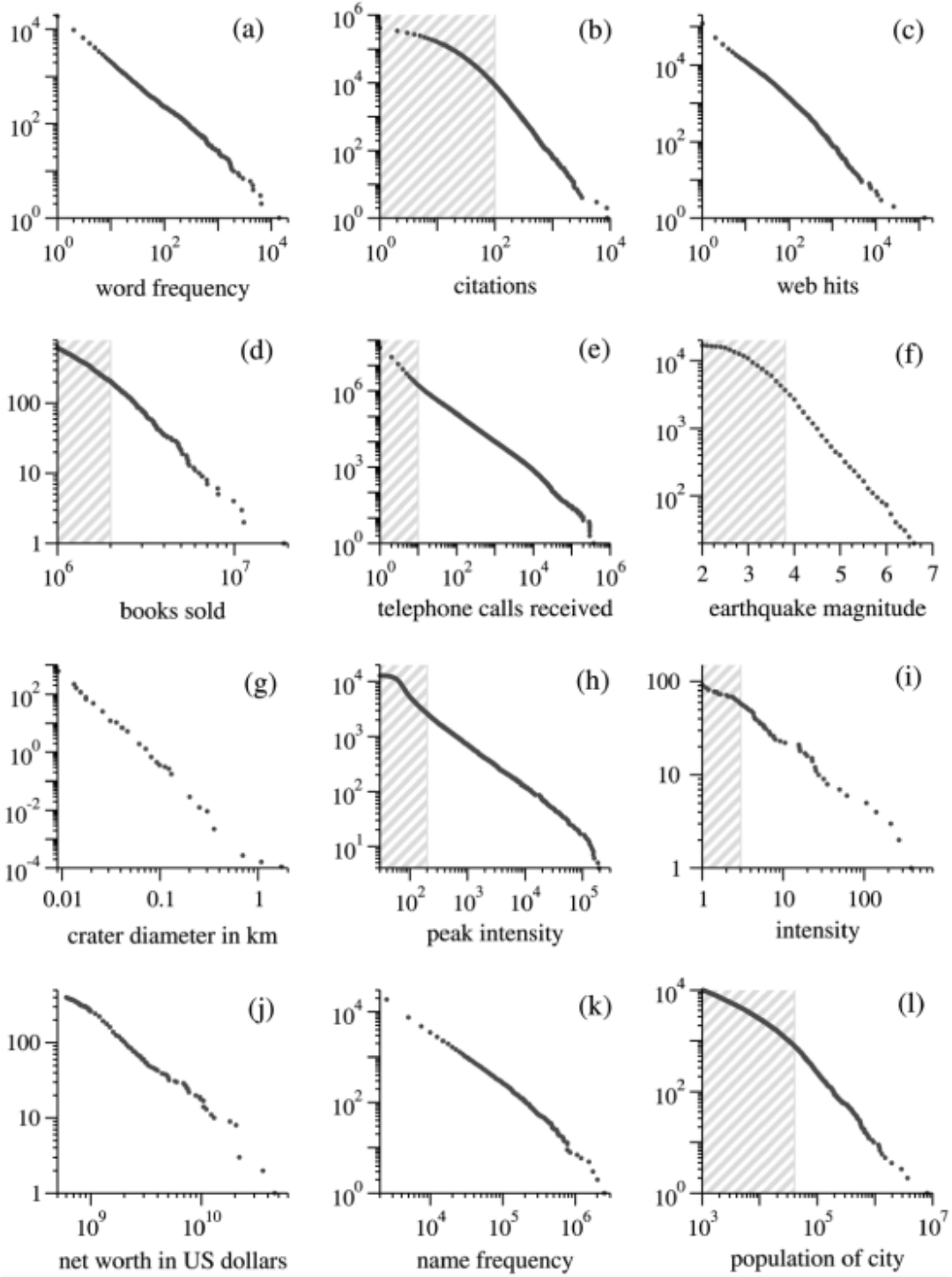



http://www.math.union.edu/~framem/IMA/FB/NatFrac/Lung1.gif

Power laws

Power laws in Nature




(a)
word frequency
(b)
citations
(c)
web hits
(d)
books sold
(e)
telephone calls received
(f)
earthquake magnitude
(g)
crater diameter in km
(h)
peak intensity
(i)
intensity
(j)
net worth in US dollars
(k)
name frequency
(l)
population of city

Some models producing power laws and fractals

- Ising model at critical point.
- Self-organised criticality and forest fire models.
- Preferential attachment.
- Diffusion limited aggregation.
- Highly optimised tolerance model.

We will look at the forest fire model in project 2 on exercise sheet 2.

See Newman (2005) for more on power laws.