11. Cellular automata

[Gould-Tobochnik 15, Tapio Rantala's notes; http://www.wolframscience.com/; http://www.stephenwolfram.com/publications/books/ca-reprint/contents.html]

11.1. Basics

Cellular automata ("soluautomaatit' in Finnish) were invented and first used by von Neumann and Ulam in 1948 to study reproduction in biology. They called the basic objects in the system cells due to the biological analogy, which lead to the name "cellular automaton". Nowadays cellular automata are used in many other fields of science as well, including physics, so the name is somewhat misleading.

The basic objects used in the models can be called both "sites" or "cells", which usually mean exactly the same thing.

The basic idea in cellular automata is to form a discrete "universe" with its own (discrete) set of rules and time determining how it behaves. Following the evolution of this, often highly simplified, universe, then hopefully enables better understanding of our own.

11.1.1. Formal definition

A more formal definition can be stated as follows.

1° There is a discrete, finite site space $\mathbf{G}$

2° There is a discrete number of states each site can have $\mathbf{Q}$

3° Time is discrete, and each new site state at time $t + 1$ is determined from the system state $\mathbf{G}$ (t)

4° The new state at $t + 1$ for each site depends only on the state at t of sites in a local neighbourhood of sites V

5° There is a rule f which determines the new state based on the old one

To make this more concrete, I'll give examples of some common values of the quantities. The site space $\mathbf{G}$ is often just a 1-dimensional or 2-dimensional set of points. The state space $\mathbf{Q}$ is often just a few integers, in the binary case simply "0" or "1". The number of possible states is often denoted with k, so for binary state spaces $k = 2$. The neighbourhood V can be just the nearest neighbours plus the state itself. For instance, in 1D it is then just "left", "same" and "right".

What the rules can be will become clear in the next section.

Note that in this definition, there is nothing really about randomness, so one might question whether CA's belong to an MC course. But the starting state and/or rule is often chosen in random, and sometimes the rule itself has randomness in it. Also, many cellular automata resemble (or are) lattice models which are also often treated with MC. Besides, CA's are sort of fun. Hence we do treat them on this MC course.

11.1.2. One-dimensional automata

The simplest one-dimensional automata are those where the state space is binary and the local neighbourhood is limited to nearest-neighbours.

The rule could be something like "if left or right is 1, new state is 1; otherwise it is zero". But a little bit of thought reveals that it is in this case quite easy to systematically consider the rules in this case.

11.1.2.1. Systematic stating of rules

In the nearest-neighbour model, there are actually only $2^3 = 8$ different configurations of neighbours and the site itself which lead to the outcome. These are:

111 110 101 100 011 010 001 000

For each of these 8 states, there are 2 possible outcomes: 0 or 1. Hence the total number of possible rules is $2^8 = 256$. For instance, the rule mentioned above would be

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

This kind of rule can be called "rule 11111010". Or if we read the string "11111010" as a binary number, we get simply "rule 250" because $128 + 64 + 32 + 16 + 8 + 0 + 2 + 0 = 250$.

This type of rule notation is the most general kind imaginable, and hence is called **general rule.**

This CA can easily be generalized. Instead of considering only nearest neighbours, one can consider neighbours up to a radius r. This strongly increases the number of possible outcomes and the number of general rules. But this can be reduced by making some reasonable assumptions. First, we assume the rule is symmetric about the middle point (i.e. for $r = 1$ $f(abc) = f(cba)$). Second, one assumes that $f(0) = 0$ for any r. Third, one assumes that the outcome only depends on the sum of the states in the neighbourhood. (i.e. for $r = 1$ e.g. $f(110) = f(101) = f(011)$).

These kinds of rules can be labeled as follows:

$$\text{rule} = \sum_m a_m 2^m$$

where m is the sum of the states in the neighbourhood (including the site itself). Then

$$a_m = 0 \quad \text{if the outcome is 0 for the sum } m$$

$$a_m = 1 \quad \text{if the outcome is 1 for the sum } m$$

The above general rule does not fulfill this stricter set of criteria: But consider now (still for $r = 1$) the following general rule:

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

This would be "rule 22" (because $2^1 + 2^2 + 2^4 = 22$) in the general rule notation. In the latter notation, we see that the possible sums are 0, 1, 2 and 3. Now we have

$$a_m = 1 \text{ for } m = 1, 0 \text{ otherwise.}$$

So this would be "rule 2" (because $2^1 = 2$) in the latter notation. This notation is called an **outer totalistic rule** (just a totalistic rule is one which does not depend on the value of the site itself). Actually, this is a restricted variety of outer totalistic rules, since it depends only on the sum of the site and its neighbours, not separately on the two. [http://www.stephenwolfram.com/publications/articles/ca/85-two/2/text.html].

Does this sound confusing? Well, it is a bit confusing, there are many different rule notations, and in reading the literature one has to make sure one knows which kind of rule is meant in each case. We will only mention general and outer totalistic rules.

11.1.2.2. Implementing the 1D $r = 1$ automaton.

Let us see what this produces. Coding something like this is quite easy of course. In the 1D examples, we can easily do without graphics, just printing the outcome of each line to standard output. Here is the code from Gould and Tobochnik (changed to C, and graphics commands removed):

```
// PROGRAM ca1
// one-dimensional Boolean cellular automata

int main();
void setrule(int update[]);
void initial(int site[], int *L, int *tmax);
void iterate(int site[], int L, int update[], int tmax);
#define MAXWIDTH 1600
int main()
{
    int L, site[MAXWIDTH], tmax, update[8], Itmp1_;

    for(Itmp1_ = 0; Itmp1_ < MAXWIDTH; ++Itmp1_)
        site[Itmp1_] = 0;

    setrule(update);
    initial(site, &L, &tmax);
    iterate(site, L, update, tmax);
}
void setrule(int update[])
```

```
{
    int bit0, bit1, bit2, i, rule;

    rule=90;
    printf("rule number = %d\n", rule);
    for(i = 7; i >= 0; --i) {
        update[i] = (rule >> i);            // find binary representation
        rule = rule - (update[i] << i);
        bit2 = i/4;
        bit1 = (i - 4*bit2)/2;
        bit0 = i - 4*bit2 - 2*bit1;
        // show possible neighborhoods
        printf("%1d %1d %1d ", bit2, bit1, bit0);
        printf("   ");
    }
    printf("\n");
    for(i = 7; i >= 0; --i) {
        printf(" %2d ", update[i]);          // print rules
        printf("     ");
    }
    printf("\n");
}
void initial(int site[], int *L, int *tmax)
{
```

```
    (*L) = 60;
    (*tmax) = 100;
    site[(*L)/2] = 1;                       // center site
}
void iterate(int site[], int L, int update[], int tmax)
{
    int i, index, sitenew[MAXWIDTH], t, Itmp1_;

    // update lattice
    // need to introduce additional array, sitenew, to temporarily
    // store values of newly updated sites
    for(t = 1; t <= tmax; ++t) {
        for(i = 1; i <= L; ++i) {
            index = 4*site[i - 1] + 2*site[i] + site[i + 1];
            sitenew[i] = update[index];
            if(sitenew[i] == 1) printf("X");
    else printf(".");
        }
printf("\n");
        for(Itmp1_ = 0; Itmp1_ < MAXWIDTH; ++Itmp1_)
            site[Itmp1_] = sitenew[Itmp1_];
        site[0] = site[L];                  // periodic boundary conditions
        site[L + 1] = site[1];
    }
```

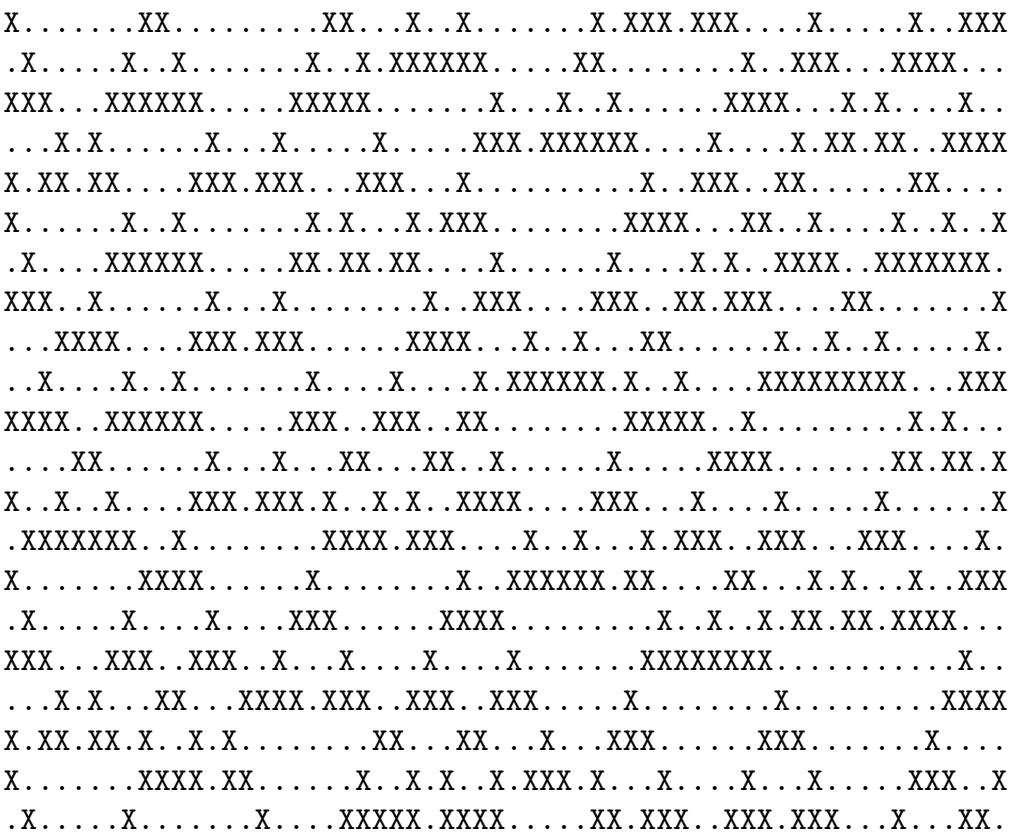
```
}
```

So here periodic boundary conditions are used, and the former notation for the rules. The program is available on the course web page (both in C and Fortran).

11.1.2.3. Outcomes of the 1D automata

If we now run this, starting from a random configuration, we get e.g. the following pattern:

```
X.XXXXXX...XXX..X.XX.....XX.X..X...X.....XXX..XXXXX...X...X.
X.......X.X...XXX...X...X...XXXXX.XXX...X...XX.....X.XXX.XX.
XX.....XX.XX.X...X.XXX.XXX.X.........X.XXX.X..X...XX........
..X...X......XX.XX.........XX.......XX.....XXXXX.X..X......X
XXXX.XXX....X.....X.......X..X.....X..X...X......XXXXX....XX
........X..XXX...XXX.....XXXXXX...XXXXXX.XXX....X.....X..X..
.......XXXX...X.X...X...X......X.X..........X..XXX...XXXXXX.
......X....X.XX.XX.XXX.XXX....XX.XX........XXXX...X.X......X
X....XXX..XX..............X..X.....X......X....X.XX.XX....XX
.X..X...XX..X............XXXXXX...XXX....XXX..XX......X..X..
XXXXXX.X..XXXX..........X......X.X...X..X...XX..X....XXXXXX.
.......XXX....X........XXX....XX.XX.XXXXXX.X..XXXX..X.......
......X...X..XXX......X...X..X............XXX....XXXX......
.....XXX.XXXX...X....XXX.XXXXXX...........X...X..X....X.....
....X........X.XXX..X..........X.........XXX.XXXXXX..XXX....
...XXX......XX....XXXX........XXX.......X..........XX...X...
..X...X....X..X..X....X......X...X.....XXX........X..X.XXX..
.XXX.XXX..XXXXXXXXX..XXX....XXX.XXX...X...X......XXXXX....X.
```



Note that even though we start from a completely random configuration, a clear fractal-like pattern soon emerges.

The 256 binary $r = 1$ CA's have actually all been examined and the behaviour analyzed, by Stephen Wolfram (the same person who is behind Mathematica) among ot-

hers. See http://www.stephenwolfram.com/publications/articles/ca/86-caappendix/3/text.html or http://www.wolframscience.com/nksonline/page-53 for a complete list.

11.1.2.4. Classification of CA's

The long-term behaviour of the binary cellular automata can be classified as follows:

1° A homogeneous state where every site is either 0 or 1 (i.e. same everywhere. This can be e.g. a self-extinguishing or self-filling pattern. For $r = 1$ the general rules "0" and "255" obviously belong here, since they either empty or fill space immediately

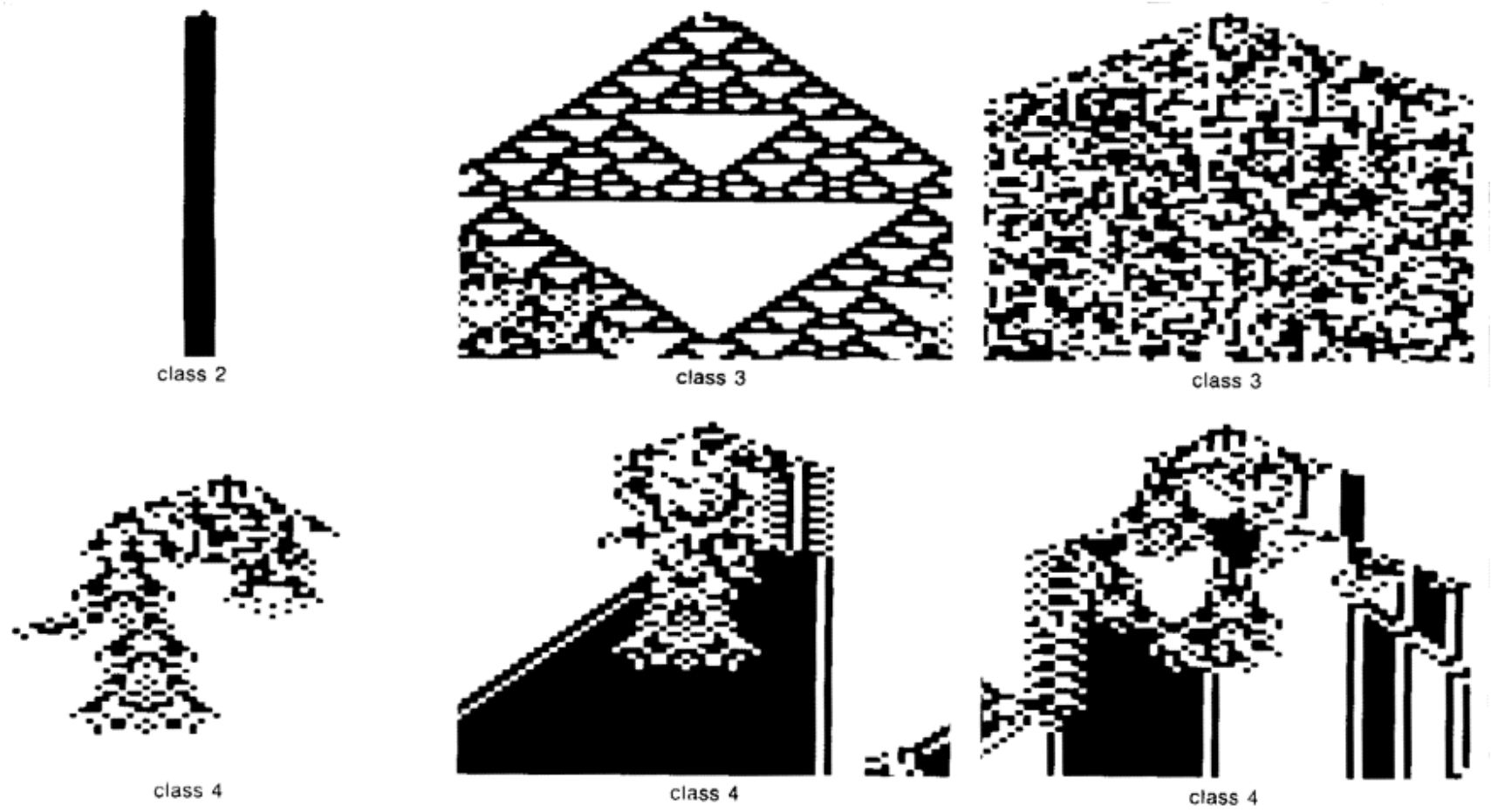
2° Patterns consisting of separate stable or regions periodic in time. In these the initial state quickly evolves into a single pattern which then stays unmodified, or switches back-and-forth between two states

3° Propagating patterns which increase in size as time advances, which are not periodic in time. These have chaotic properties, i.e. the prediction of the value of a site at infinite time would require knowledge of an infinite number of initial site values.

4° Complex, localized structures which may take on very complex shapes, but may not exist forever.

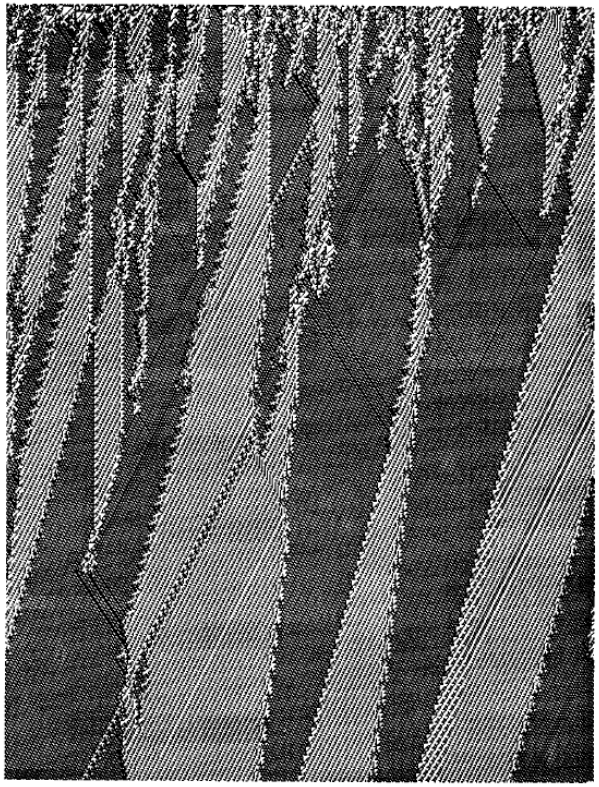
For $r = 1$ there are no automata of the most interesting type 4. For $r = 2$ there are, as you will find in the exercises.

Here are a few examples clarifying what classes 2-4 can be like:



It turns out that even for these seemingly trivial 1D automata, some of the class 4 automata may take on amazingly complex shapes (see figure below).

In fact the basic $k = 2, r = 1$ rule 110 has so called universal computing behaviour [http://www.wolframscience.com/nksonline/page-676-text]. What this means will be discussed for 2D automata in the next section.



11.1.3. Two-dimensional automata

Even though complex shapes do emerge from the 1D automata, they seldom have true aspects of evolution and self-organization, and hence are finally not that interesting in terms of studying complexity. This changes completely when we go up to 2 dimensions.

In two dimensions, the number of possible automata of course goes up into huge numbers. But if we only consider nearest-neighbour automata, and only use the totalistic sum rule, the number of automata to consider remains manageable.

These automata can be described with rules just as the 1D ones. The general rule for outer totalistic automata with k states is

$$\bar{C} = \sum_{a=0}^{k-1} \sum_{m=0}^{n_n} \bar{f}(a,m) k^{km+a}$$

where m is again the sum of the states in the neighbourhood, now excluding the site itself. n_n is the total number of possible sites in the neighbourhood. $f(a,m)$ is the new state of the site and a loops over the previous states. Note that by introducing the state of the site itself into f it is possible to have different outcomes depending on whether the site was previously on or off.

11.1.3.1. The game of life

The by far most famous cellular automaton is the "Game of life". It was originally devised by John Conway in 1970, more or less as a solitaire game. However, it turned out to produce amazingly complex patterns, and has since then been the focus of much serious research (although it certainly has also been the focus of countless high-school kids fooling around with a computer).

The rule of the game are simple.

Consider a quadratic board where every square can be alive or dead (1 or 0). The next state is determined from the state of the 8 neighbours as follows.

1° If the site is dead:

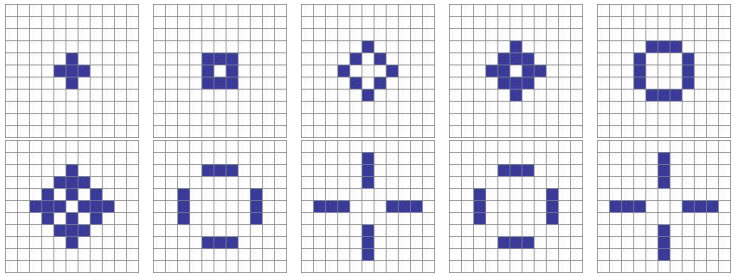
- **1 a°** It becomes alive if it has exactly 3 live neighbours
- **1 b°** Otherwise it stays dead

2° If the site is alive:

- **2 a°** It dies due to isolation if it has less than 2 live neighbours
- **2 b°** It stays alive if it has exactly 2 or 3 live neighbours
- **2 c°** It dies due to overcrowding if it has more than 3 live neighbours

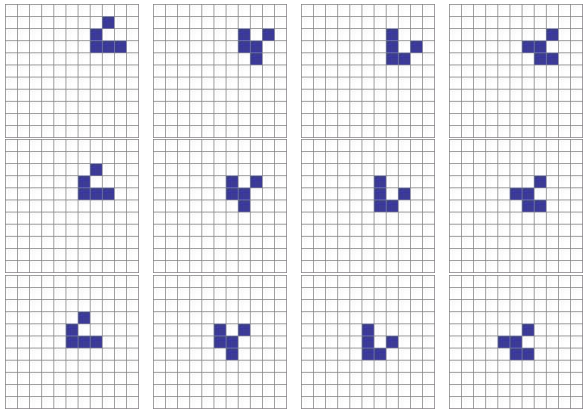
These rules correspond to $\bar{C} = 224$, as you get to prove in the exercises.

I will just give two simple examples of the outcome here. One is how four "blinkers" evolve starting from a cross:



Can you follow how this outcome derives from the rules?

Another classic is the walker:

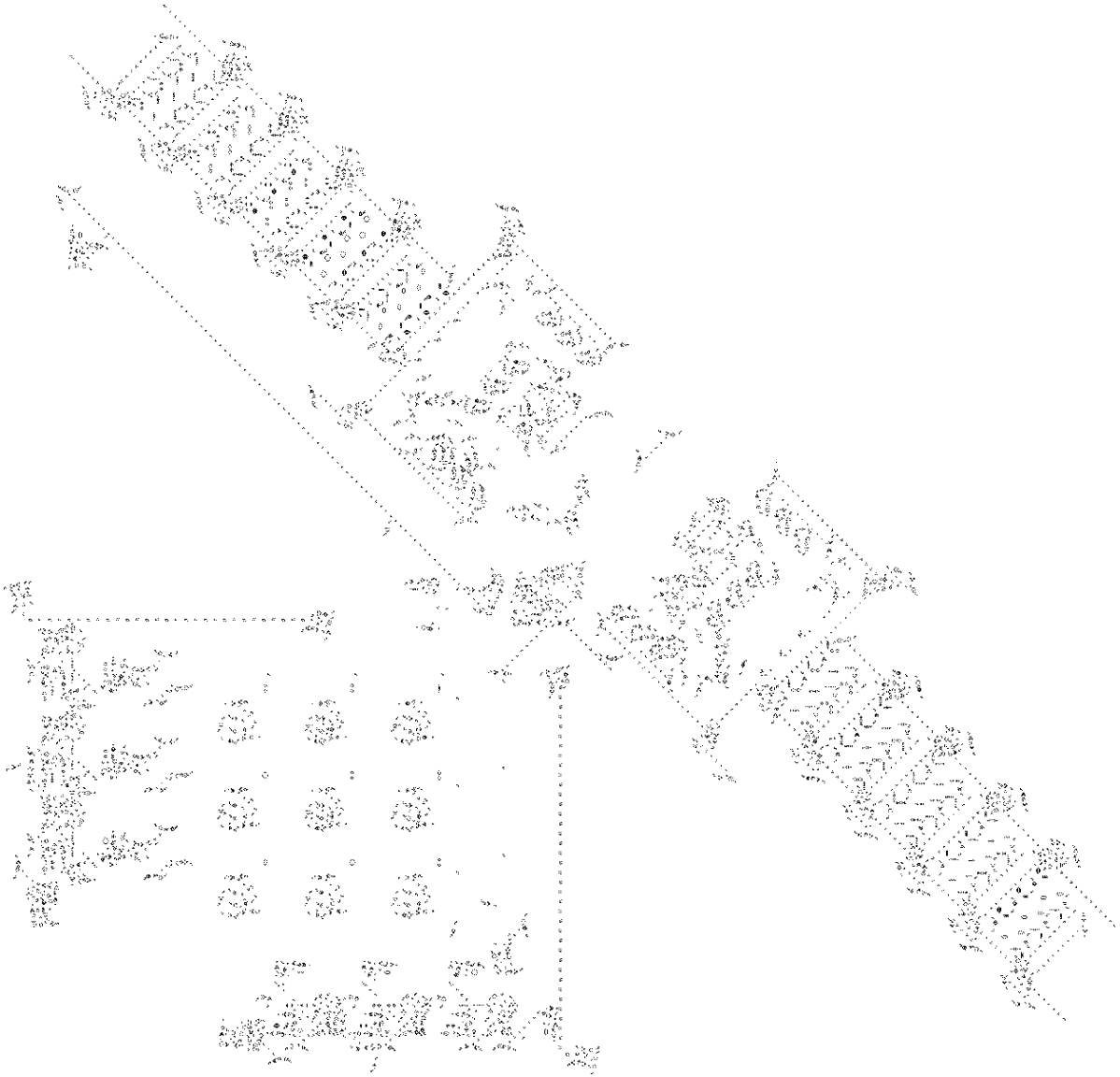


So you see that this thing keeps on returning to its original shape, but has moved one step in x and another in y before doing so.

If you like to see more of this stuff, you can play around with the java applet in http://hensel.lifepatterns.net/. The "A-plus" starting pattern shows a wide range of objects, like a wire creator, a walker creator, objects moving at different speeds, things exchanging objects, an immobile sink of some of the moving objects and so on.

The most amazing feature of the "Game of life" is that it has been shown that all of the basic components of a computer can be constructed in it: analogies to wires, memory, and the logical circuits (AND, OR etc.). This means that a "universal computer" (Turing machine) can be constructed in the game of life. Since precent-day computers are based on the ideas of Turing machine, this means that the logic circuits of any present-day computer could be reproduced in the game of life.

This was actually shown long ago, in 1982 or before, by showing how all the individual components could be made up. In 2000, Paul Rendell actually constructed such a machine, a working Turing machine with 3 states and 3 symbols. Here is the starting design:



This design takes 11040 life generations to do a single cycle of the computer. So from a practical viewpoint this thing is not exactly a very efficient way to build a computer. Still, as a concept it is quite fascinating.

There have actually been real computers which are based on cellular automata. In the 1980's Thinking machines corp. built the "Connection Machine" which was based on 1-bit processors which where physically connected to form a cellular automaton. It was also one of the first commercial parallel computers. By now the company has gone bankrupt, which shows that cutting-edge research does not necessarily lead to good business...

Maybe the greatest importance of this kind of cellular automata is that they nicely illustrate how some extremely simple rules (like the ones in the game of life) can be used to produce a system with very complex behaviour. From a physics philosophy point of view, this gives some hope for us physicists that some truly simple and elegant "theory of everything" describing the whole universe does actually exist and could one day be found.

Stephen Wolfram actually thinks that the cellular automata framework of thinking could be the path which leads to the theory of everything, but this idea is not exactly shared by the wider physics community... He also argues that reality is often so complicated that often the best tool to describe it are computer simulations and not analytical mathematics. For 1D automata this can be exemplified by the fact that it is impossible to predict the exact state of 1D automata after N

iterations without running the whole simulation. This notion has in part been disputed in [N. Israeli and N. Goldenfeld Phys. Rev. Lett. 92, 074105], who showed that it is possible to do an analytical prediction of the overall state of 1D automata, where the details are fuzzed out. They argue that in many physical systems the details are unimportant in any case, hence justifying the use of analytical mathematics. [http://focus.aps.org/story/v13/st10]

11.2. Physical applications

As has become clear from the above, much of the interest to cellular automata comes from computer science and the study of complex phenomena and chaos. But there are important applications in physics as well, some of which we will now mention.

11.2.1. The Ising model

Recall that the Ising model describes the energy of a spin system as a sum of nearest-neighbour spins,

$$E = -J \sum_{i,j=nn(i)}^{N} s_i s_j \tag{1}$$

We have already heard during this course that the Ising model can be simulated using KMC and Metropolis MC. Well, it turns out that it can further be simulated as a cellular automaton as well.

The basic relation with cellular automata is clear: the 2D lattice forms directly the CA set of cells. Now the neighbours considered are the 4 nearest neighbours.

In the CA application of the Ising model, the energy of the entire lattice is fixed, and a spin flips only if the total energy is not changed. This is the case if a spin has 2 up and 2 down neighbours: then a spin flip will not change the total energy. In this case, a flip is always performed.

In CA, we always want to update all neighbours simultaneously. This leads to a problem: if two neighbouring spins of opposite sign both have a total of 2 up neighbours, then switching the spin of these two would lead to a change in total energy.

To overcome this problem, one can divide the board into a chessboard pattern. Then one first updates the white squares all simultaneously, and after this the black squares. Since no white and black square on a chess board ever are adjacent, this will overcome the problem mentioned above.

An interesting feature about this CA implementation of the Ising model is that it is not ergodic: there are some states with the same energy which will never be reached for a given initial condition. This can be seen e.g. by considering the minimal 2×2 lattice with periodic boundaries. Arrange the spins as

$$\begin{array}{cc} + & - \\ + & - \end{array}$$

and the system will, when we first flip $1/2$ of the states, go to

$$\begin{array}{cc} - & - \\ + & + \end{array}$$

and then to

$$\begin{array}{cc} - & + \\ - & + \end{array}$$

which is the end result for the first update. The next sequence will return to the original state. Hence the state

$$\begin{array}{cc} + & + \\ - & - \end{array}$$

which obviously has the same energy as the original one, will never be reached (unless we count the intermediate 'half' states, which we should not).

To overcome the ergodicity problem, one can e.g. bend the rules a bit. Allow with a small probability a randomly chosen spin to swap state regardless of its neighbourhood, selecting the probabilities so that the total energy will still on average be the desired E_0. Hence we still effectively conserve the energy E within some small fluctuation interval δE, but overcome the barrier problem.

11.2.2. Modeling fluid flow

[GT 15.2. But see also e.g. http://www.santafe.edu/ shalizi/reviews/rothman-zaleski-on-lga/]

One of the most important applications of CA's in physics today is fluid flow. Simulating fluid flow is very difficult because the basic equation governing it, the Navier-Stokes differential equation, is nonlinear. Moreover, in practical simulations one often wants to deal with systems which have many length scales, and involve complex shapes with obstacles etc. All of this makes direct solution of the equations complicated.

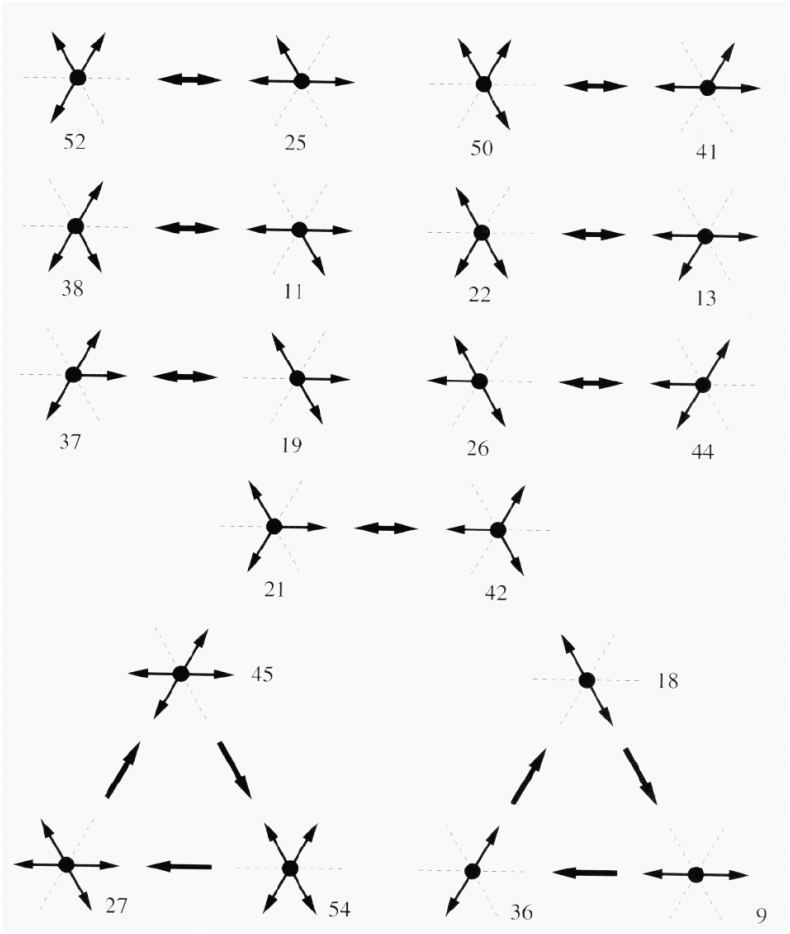
Cellular automata offer another route to simulating fluid flow. The CA model for fluid flow is a **lattice gas** model. This means that the particles are restricted to the site of a lattice, and the velocities are also restricted to a small number of velocity vectors.

For fluid flow, the CA modeling starts from the idea that if we maintain the conservation laws and symmetries associated with the fluids in the microscopic, simplified picture, then one can reproduce the correct macroscopic physics as an average of a large number of microscopic particles.

In the CA model, a triangular lattice is used, since this can describe the continuum symmetry, which a simple square model could not. The possible velocity vectors are the 6 vectors which link nearest-neighbour sites in the lattice. In the model we describe, rest particles are not allowed. A site

in the lattice can be occupied by 0-6 particles, so that the site can contain at most one particle moving in a given direction.

The simulation proceeds by first moving the particles by their current velocity vector. Then it considers all particles at a given site, imagining that this is a particle collision. A set of predetermined rules tells what the outcome (i.e. the new velocity vectors) are after a collision. These rules are formulated so that they fill the required conservation laws. The possible collisions and their outcomes can be graphically illustrated as follows:



What does this mean? For instance, in the left upper corner the arrows linking 52 and 25 show that state 52 goes to state 25 on collision, and vice versa. The numbers refer to a bitwise encoding of the possible collisions, which we will not present here (see Gould-Tobochnik if interested). In all

cases we see that the number of particles in and out of a state is conserved, which means that we do have mass conservation. That the system otherwise behaves as a fluid is not as obvious, but has been proven (by Uriel Frish, Brosl Hasslacher, and Yves Pomeau in 1985, and independently somewhat later by Stephen Wolfram, who even $patented$ the whole concept of lattice gases).

One great advantage with the approach is that encoding one point in space requires at minimum only 6 bits of information, much less than in any other method. This enables using very large systems. On the other hand, since macroscopically meaningful information can be obtained only as the average of a large number of points, a large system is always needed.

For an example on how this behaves on a larger scale see http://www.wolframscience.com/nksonline/page-378 and on

---

How useful has this really been? Since I am no expert myself on fluids, I give a direct citation from an expert in the field [http://www.santafe.edu/ shalizi/reviews/rothman-zaleski-on-lga/]

"Since the mid-1980s, however, the limitations of lattice gases have become clearer, and early hopes that they would let us crack the problem of turbulence, for example, have been dashed.

Lattice gas have by no means however proven useless; in fact, they have many applications, especially

to situations where we face messy boundary conditions, irregular obstacles in the way of the flow, or more than one type of fluid. This covers a lot; in particular, the petroleum industry is very interested in how to force oil out of rock. But there are many other important applications, too, having to do with pattern formation, and even to straight-forward fluid flow when the Reynolds number (a kind of ratio of inertia to viscosity) is not too large. "

11.2.3. Self-organized critical phenomena

Consider very large-scale events which occur very rarely. Such events can be e.g. earthquakes, avalanches, or a stock-market crash.

Large-scale events could be triggered by some very strong forces, or special out-of-the-ordinary events which normally never occur. However, it could be they are also triggered by some very ordinary minor event which happens all the time, but sometimes triggers a large-scale effect for some reason. The study of **self-organized criticality** deals with trying to understand the latter variety of events.

Out of the examples mentioned above, we know that at least avalanches and stock-market crashes can be triggered by both kinds of events. An avalanche can be triggered intentionally by e.g. explosives placed in the right spot, and the (minor) stock-market plunge after Sep 11, 2002, clearly was triggered by the actions of a few ruthless terrorists/valiant freedom fighters/misunderstood youths (Western/Arab nationalist/Swedish social democratic world view). On the other hand, we know that some avalanches start completely on their own with no major disturbance. For instance, no single clear-cut out-of-the-ordinary trigger for the stock-market plunge of 1987 has been found.

More scientifically, a system is said to be **critical** if the number of events $N(s)$ follows a power law:

$$N(s) \propto s^{-\alpha} \tag{2}$$

Compare this to the result of combining a large number of independently acting events (e.g. the random walks discussed earlier on during this course). The central limit theorem shows that the probability distribution for an event to occur is usually a Gaussian, i.e.

$$N(s) \propto e^{-(s/s_0)^2}$$

These two kinds of behaviour differ in that the critical behaviour does not have a characteristic length scale, i.e. is **scale invariant.** This means that if we change the length scale by $s \longrightarrow bs$, we get

$$N'(s) \propto (bs)^{-\alpha} \propto b^{-\alpha} s^{-\alpha} \propto s^{-\alpha}$$

i.e. only the prefactor changes, but not the slope of the function. In the Gaussian on the other hand, plugging in bs changes the s_0 factor in the exponential to s_0/b, i.e. s_0 gives a characteristic length scale.

A practical important difference is that because an exponential form always approaches zero faster

than any power law with a constant exponent α, a Gaussian form gives virtually no events for large s, whereas the power law tends to give a few even for very large s.

11.2.3.1. Simulation of a sand pile

The perhaps most commonly cited example of a self-organized critical phenomenon is a sand pile. Consider grains of sand falling on a flat square with open edges. Initially the grains which land, will stay at their initial positions. But as the pile keeps growing, the system will end up with steep, unstable slopes which will collapse downwards in small sand avalanches. From our everyday experience we know that the sand will eventually reach a steady-state shape close to that of a cone, which is stabilized by small avalanches which take away the additional grains which fall in.

This system is critical because (once steady-state is reached) there will be avalanches of all sizes, up to the whole size of the system. The system is also self-organized, because no kind of processing (temperature, pressure or the like) is needed to get the system to its steady-state configuration.

Doing sand pile experiments is of course easy, every one of us has done that in our childhood. But in case we want to obtain a deeper understanding of the avalanching phenomena, how should we deal with this situation theoretically? This variety of problems (which has many other examples with clearer economic interest) has not been studied long, and is in fact very difficult to deal with

analytically. Cellular automata have turned out to be extremely useful for studying this kind of problem, since quite simple rules often can lead to the right kind of behaviour.

A maximally simple model of a sand pile is as follows.

Consider a 1D system with L sites $1, \ldots, L$. Let the height of each site be $h(i)$.

1° Add a grain to the left-most site: $h(i) = h(i) + 1$

  2° For all sites i, if $h(i) - h(i+1) > 1$ mark the site for toppling

  3° Topple all marked sites to the right using $h(i) = h(i) - 1$ and $h(i+1) = h(i+1) + 1$

  4° Set h(L+1)=0, i.e grains beyond L are lost forever

  5° If any sites were toppled, return to step 2

6° Return to step 1

Note that the sand keeps toppling until a steady configuration is reached at each time step, which is intuitively a fairly natural behaviour.

This model here is actually not yet very interesting: after it has reached the steady state (which is

just a pile with exactly the angle 45°), all avalanches will be exactly 24 steps in length, and simply lead to the single added particle leaving the system at the edge.

But it is easy to generalize this into a system with quite interesting behaviour. First, we change the toppling system as follows:

2° For all sites i, if $h(i) - h(i+1) > 2$ mark the site for toppling

3° Topple all marked sites two steps to the right using $h(i) = h(i) - 2$ and $h(i+1) = h(i+1) + 1$, h(i+2)=h(i+2)+1

Using this change the system will already exhibit a power-law dependency.

We can in addition make the system more natural by allowing the toppling to occur both to the left and right, and dropping grains randomly everywhere in the interval $[1, L]$. This introduces only one complication: what should we do if the number of grains is more than 2 larger than both left or right, but only 3 larger than either of them? Then it is not sensible to topple both left and right, since there are not enough grains in total (one grain in the middle bin would move upwards, which is clearly undesirable). A natural solution is to choose one of the topple directions randomly, and then topple only 2 grains.

This model will give a power-law dependence for same range in the toppling size, as you will see in the exercises for yourself.

Here are a few examples of how the system proceeds (being lazy, I implemented the code only with ascii graphics), for $L = 20$: The line just shows where the holder of the pile is standing. The numbers at the end of the line give some information about where we are going: `hsum` is the total number of grains in the system, the other names are self-explanatory.

```
               X
--------------------          nstep 1 ntopple 0 hsum 1
          X    X
--------------------          nstep 2 ntopple 0 hsum 2
        X X    X
--------------------          nstep 3 ntopple 0 hsum 3
        X XX   X
--------------------          nstep 4 ntopple 0 hsum 4
   X    X XX   X
--------------------          nstep 5 ntopple 0 hsum 5
```

So in the very beginning nothing much of interest happens. But if we look forward we will start to see toppling events, consider e.g. the following two steps:

```
            X
      X     X
     XXXX  XXX
   XXXXXXXXXXX
```

```
   XXXXXXXXXXXXXX
 XXXXXXXXXXXXXXXX
 XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXX
--------------------          nstep 148 ntopple 2 hsum 109
       X
      XXXX  XXX
   XXXXXXXXXXXX
   XXXXXXXXXXXXXX
 XXXXXXXXXXXXXXXX
 XXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXX
--------------------          nstep 149 ntopple 14 hsum 107
```

On the right side a clear collapse has occurred, reducing the system size by 2.

The steady state pyramidal shape is reached much later. During that, most of the time not much happens, just as predicted by a critical power law, but sometimes large events do occur. Here is an example:

```
         X
        XX
       XXXX
      XXXXX
      XXXXXX
```

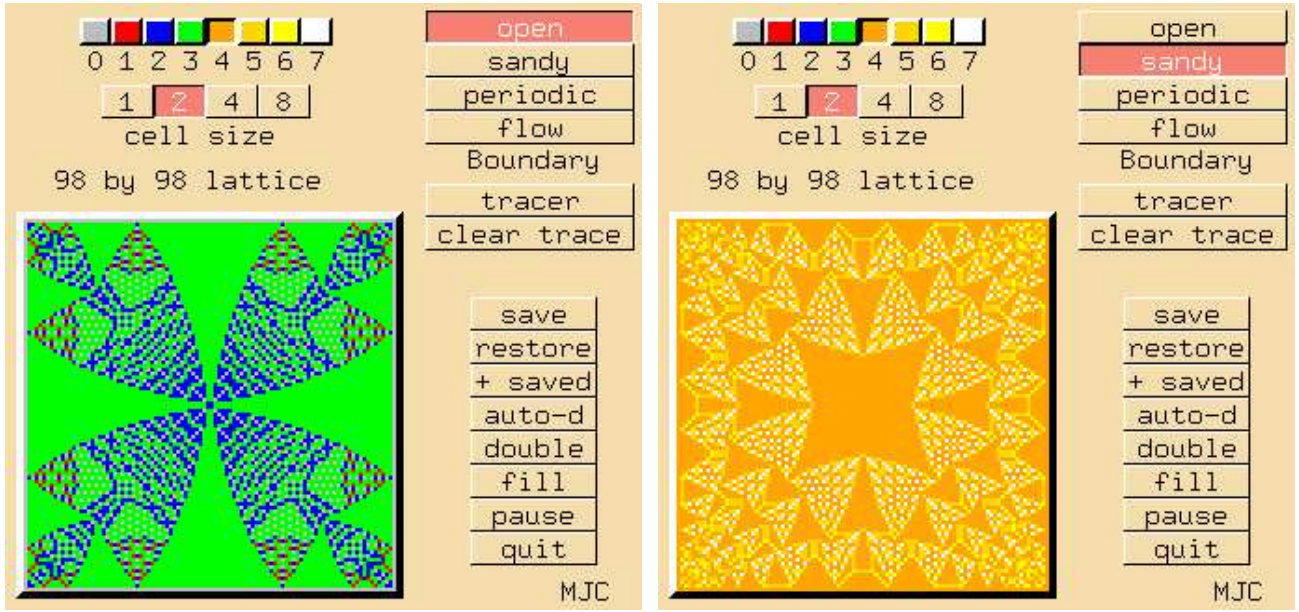
So here 100 grains have moved in one step, and 12 grains left the system.

In most coding and treatment of sand-pile like simulations, people do not actually use $h(i)$ as the variable. Instead, they use the relative difference of one site to the next, a.k.a. the **local slope.** In 1D this can be simply

$$m(i) = h(i+1) - h(i)$$

For the simple model given in the algorithm box, the criterion for toppling then simply becomes $m(i) > 1$.

The sand pile simulations can easily be generalized to 2 dimensions. The rule could be for instance that if $m(i) > 3$ one grain is given to each of the four nearest neighbours of a site. [Bak, PRL 59 (4) 381]. In 2D the patterns which form on the pile can be very interesting and beautiful; here are a couple of examples of steady states obtained with different models (these are from the xsand program available in the xtoys package; a link can be found at the course web page):



But the basic idea of the scaling law already becomes apparent from the 1D model, so I will not say more about the 2D models.

How realistic are these models? The fact that they produce power laws for the avalanche size has indeed been observed in real sand piles, but only when the pile is small. For modeling larger piles more sophisticated models have to be used.

11.2.4. Earthquakes

The dependence between an earthquake frequency of occurring and its energy release has been found to obey a power law of the type

$$N(E) \propto E^{-b}; \qquad b \approx 0.5$$

which is known as the Gutenberg-Richter law.

Earthquakes can of course be modeled by actually trying to make a model of a fault line between tectonic plates, with a system of coupled masses separated by rough surfaces. This is (not surprisingly) very expensive to do computationally.

A simple cellular automaton can, however, describe some of the basic physics of this system. Define a real variable $F(i, j)$ on a square lattice of size L^2. F represents the force on the block at position (i, j). The force matrix is initialized to some small random forces. The simulation loop then works as follows:

1° Increase F everywhere by a small constant amount ΔF, e.g. 10^{-5}, which corresponds to the force created by the slow movement of the tectonic plate

2° For all (i, j) check whether $F(i, j) > F_c$, where F_c is the critical force. For convenience, we set $F_c = 4$. If it is:

3° Release force due to 'slippage' by doing:

$$F(i, j) = F(i, j) - 4$$

and

$$F(i', j') = F(i', j') + 1 \quad \text{for the 4 nearest neighbours } (i', j') \text{ of } (i, j)$$

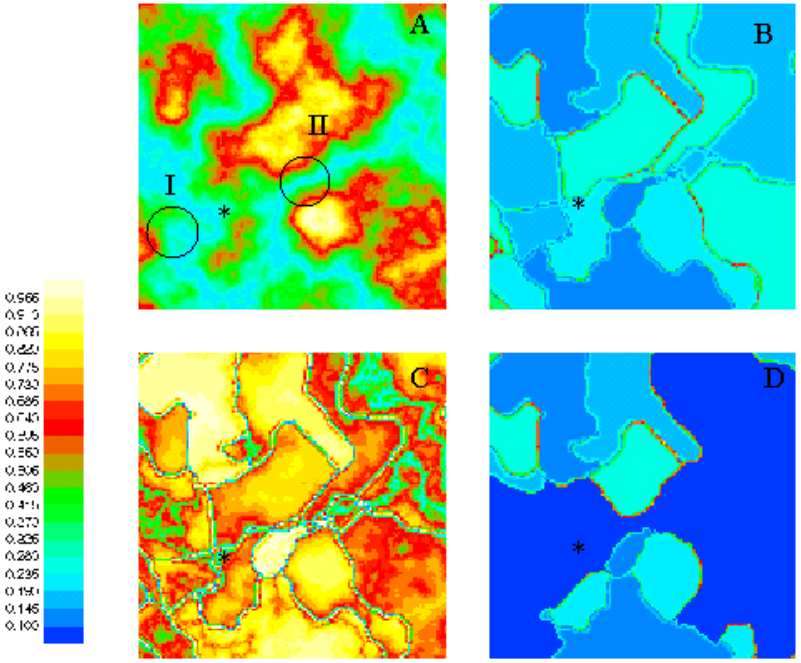
4° Return to step 1

When this is simulated, the system will eventually reach a steady state where the average force does not change any more. One can then monitor the distribution of the number of blocks s which are affected by an instability at the same time. This turns out to have a dependence close to the real one.

(Actually, something seems to be missing from this implementation given in Gould-Tobochnik: in this way the forces would grow forever. Or maybe the force is vanishing at the edges. Unfortunately I could not find easily an alternative source or code where to check this).

More advanced CA models seem to be in wide and serious use in geophysics. Papers on the topic are published by geophysics groups and in geophysical journals, which indicates it is just not a bunch of college kids playing around with fancy animations.

Here is an example of what fault lines from a CA simulation can look like:



Spatial distribution of A) strength, B) stress and C) stress deficit just before large event that starts at the asterisk. Low values of stress deficit indicate areas which are near failure. The state of stress after the event is shown in D) where the dark blue area represents the rupture.

11.2.5. Forest fire models

Simulating fires (bush, forest, and fires in a 3D space) is a popular application for cellular automata. Again we will give just one simple example. But slightly more advanced methods are in widespread use.

A simple forest fire model can be constructed as follows. We start again with an L^2 grid. The simulation then proceeds as follows.

0° Fill the sites randomly with trees with a probability p_t

1° Loop over all sites (i, j):

- **2 a°** If the site has a tree, it catches fire with probability f
- **2 b°** An empty site grows a tree with probability p
- **2 c°** If a site with a tree has a neighbour on fire, it catches fire
- **2 d°** If a site is on fire, it dies and becomes empty

3° Return to step 1

Note that steps 2 a - 2 d can either occur synchronously (i.e. at the same time), or in sequence after each other. The latter approach is used in the xfires.c program part of the xtoys package: this is the animation which has been running on the course web page all spring!

There are many fairly obvious ways in which this simulation could be made more realistic. The first is that the spreading of the fire could be made instantaneous, i.e. to occur all in one time step. Because a forest fire in reality occurs on a much shorter time scale than tree growth, this would correspond better to a sensible time scale.

Another improvement could be to make the probability of ignition proportional to the density of the trees in the forest.

The points made in the last two paragraphs I thought of myself. Then doing a web literature search, I found that both indeed have been taken into account in modeling of bush fires in Australia [www.csu.edu.au/ci/vol08/li01/li01.pdf; published in Complexity International vol. 8]. So have more complex factors such as land height, flammability, and wind conditions.

Here is a figure from the publication mentioned, showing where fires occur in a system with variable land height:

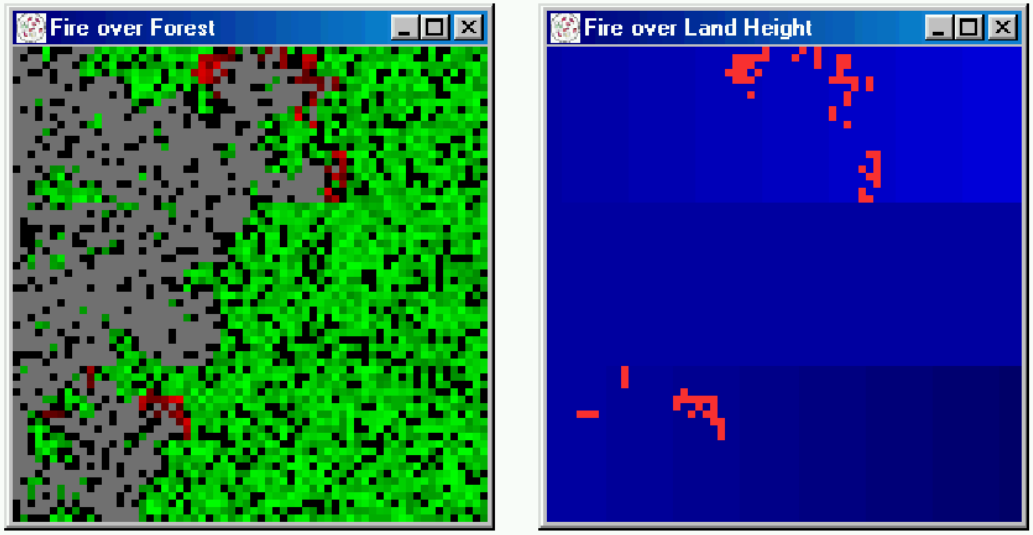


Figure 5. A forest divided into three sections: a ramp sloping upwards and eastward, a flat middle section and a ramp sloping down and to the east. a) shows the grey burnt trees over the green vegetation whilst b) shows the height of the land (brighter blue is higher). Both are overlaid with red spots denoting fire.