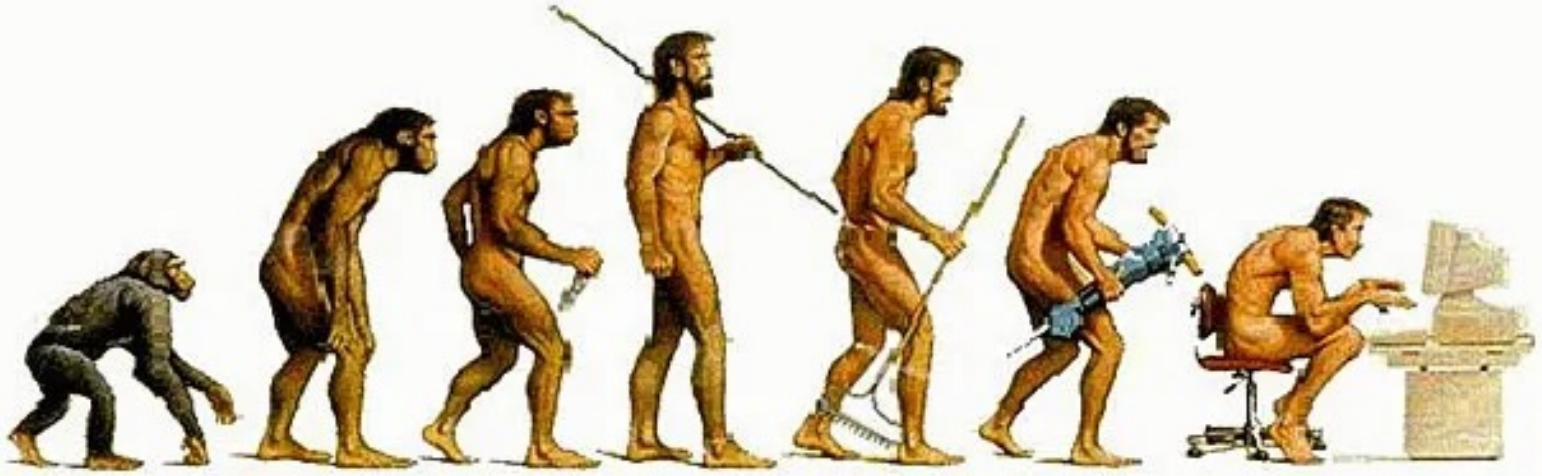
Computer modeling of physical phenomena



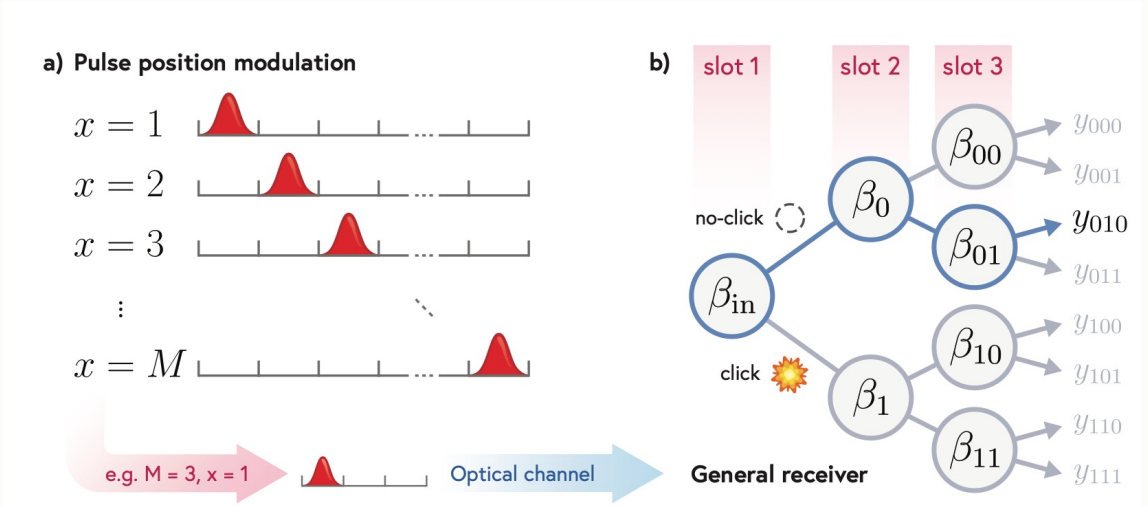
LECTURE V: EVOLUTION AND GENETIC ALGORITHMS

contact/consultation: k.lukanowski@uw.edu.pl

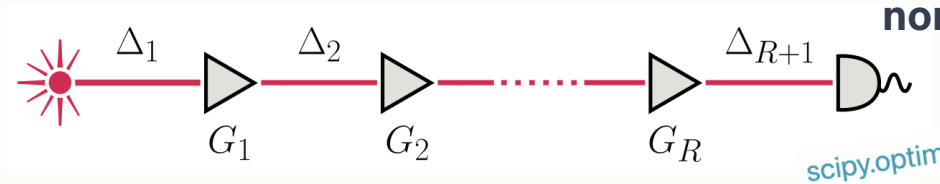
webpage stays the same: fuw.edu.pl/~tszawello/cmpp2024

but the animal kingdom is taken over

my daily work: **numerical optimization** of quantum-enhanced optical communication

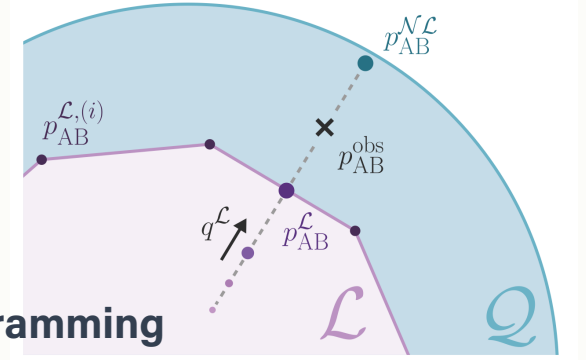


optical receiver algos



nonlinear optimization

scipy.optimize.differential_evolution



convex programming

evolution in action: ANTIBIOTIC RESISTANCE

bacterial infection!

takes antibiotics

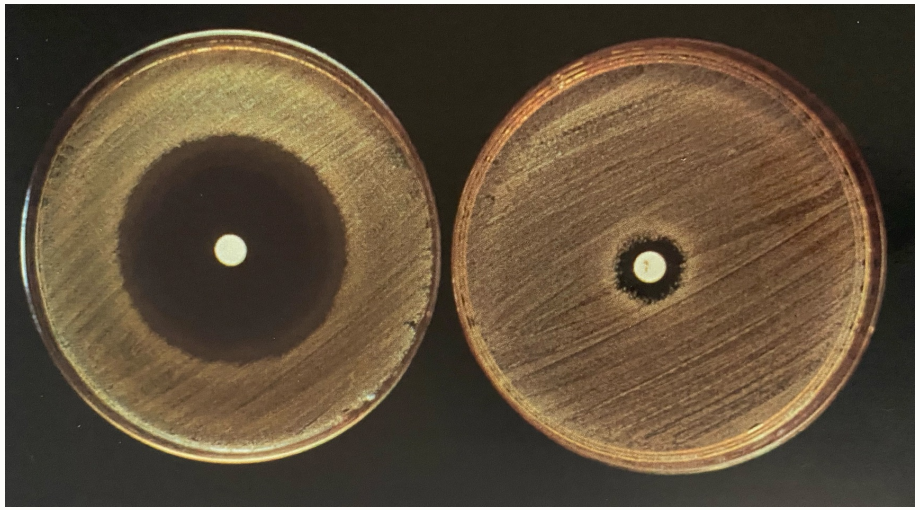
most bacteria killed, symptoms disappear :)

but the fittest survive!

stops treatment too early

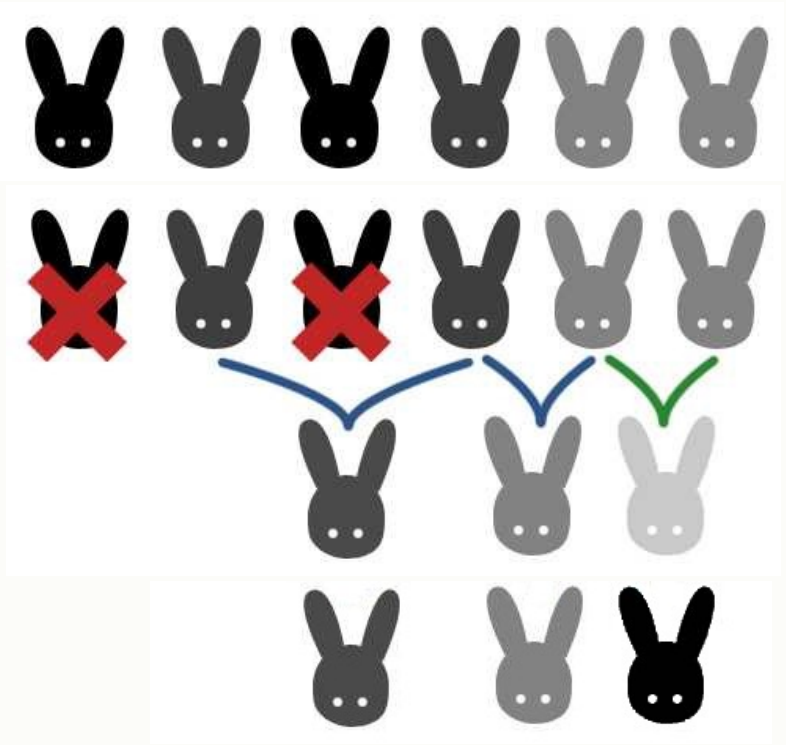
resistant bacteria spread

„superbugs"



staphylococcus aureus
vs penicillin, then and now

evolution in short



is a consequence of

- heredity (memory)
- variability (mutations)
- each generation providing more individuals than can survive (surplus)

natural selection

"survivors survive, reproduce and *therefore* propagate any *heritable* characters which have affected their survival and reproductive success" – it's not *just* "survival of the fittest"

evolution is unique

- acts on a very small subset of all possible individuals
- new variants arise by small variations of what is already present

⬇

if life started again,
the outcome would be
enitrely different

and very successful

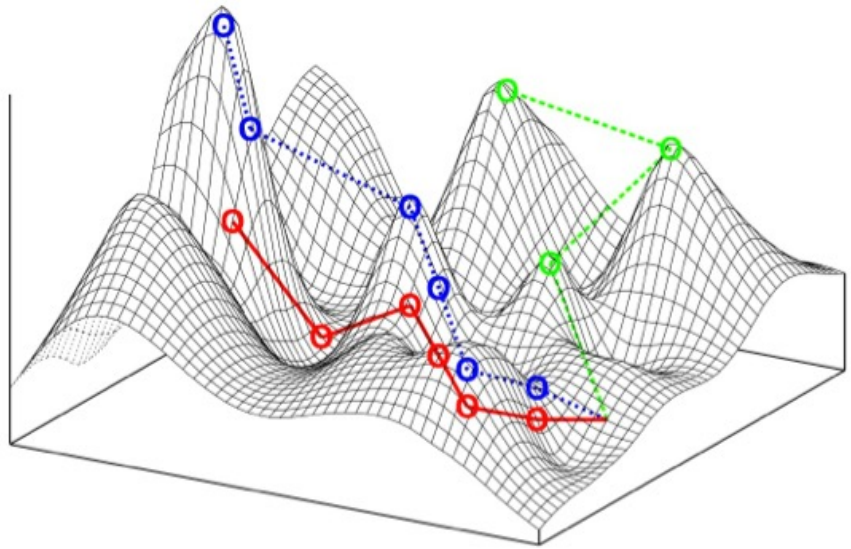
in finding (nearly) optimal
solutions to complex problems

evolutionary computation mimics that

particularly useful in cases when the complexity of the problem makes it **impossible to search for every possible solution or combination**

goal: find a good (nearly optimal) solution in an acceptable timescale within a high-dimensional, rugged fitness landscape

metaheuristic: there is no guarantee that the best solutions can be found; we don't know whether an algorithm will work and why (if it does work)

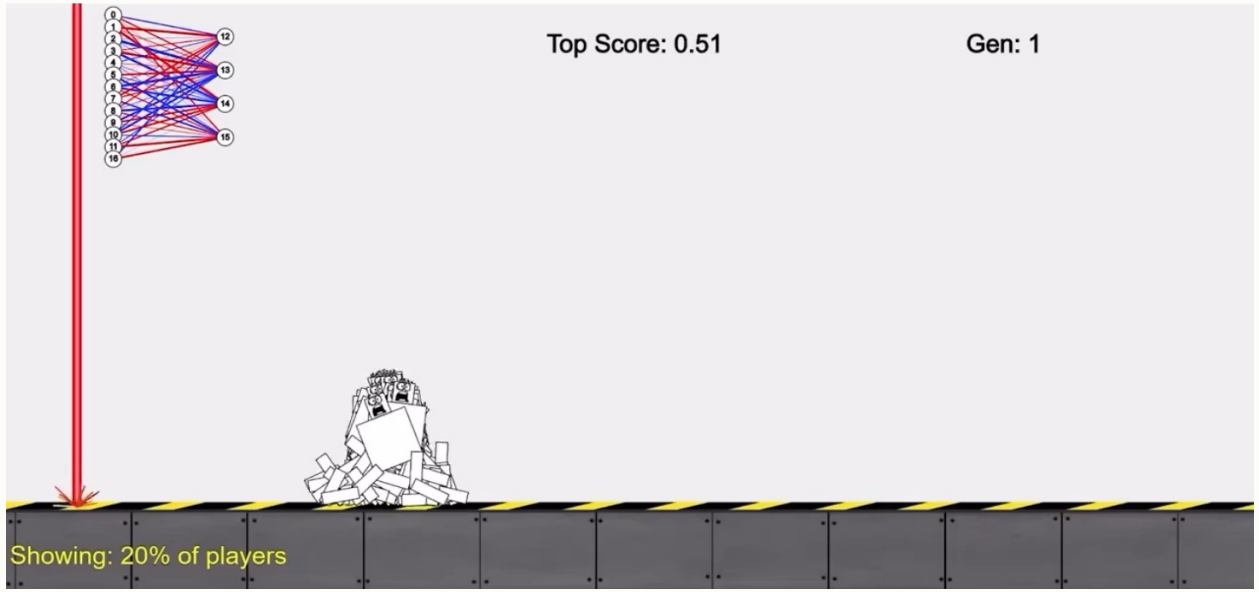


genetic algorithms: a subset of evolutionary computation

genetic algorithm

A computer simulation in which a **population** of abstract representations (called **chromosomes** / genotypes / genomes) of candidate solutions to an optimization problem (called individuals or phenotypes) **evolves** toward better solutions.

Chromosomes can be represented in various encodings, e.g. as a string of 1s and 0s.

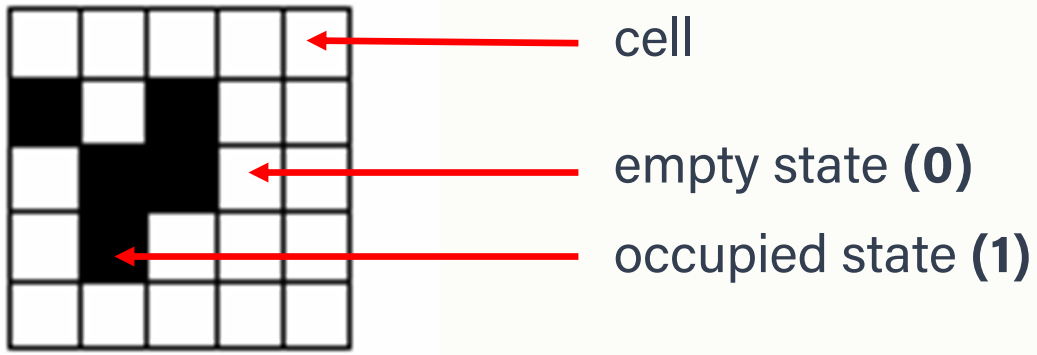


Evolution usually starts from a population of **randomly generated individuals** and happens in **generations**.

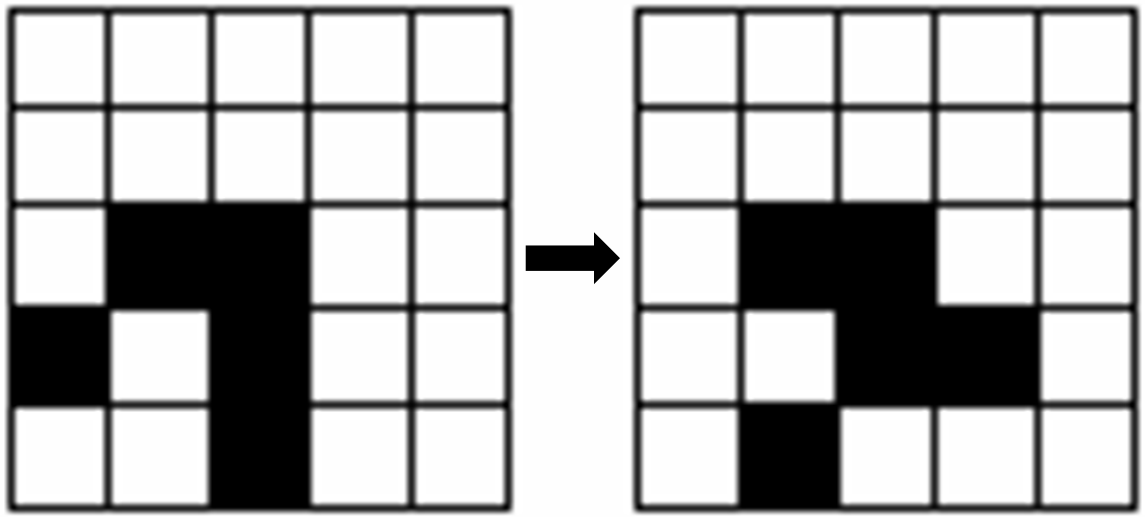
In each generation, the **fitness** of every individual in the population is evaluated. A portion of individuals is **selected** from the current population, based on their fitness. They are then modified (**recombined** and possibly **mutated**) to form a new population. The new population is then used in the next iteration of the algorithm.

Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population.

our old friends, square grids and cellular automata



Automaton defined on a grid of cells, each in one state from a discrete set (here 0 and 1)

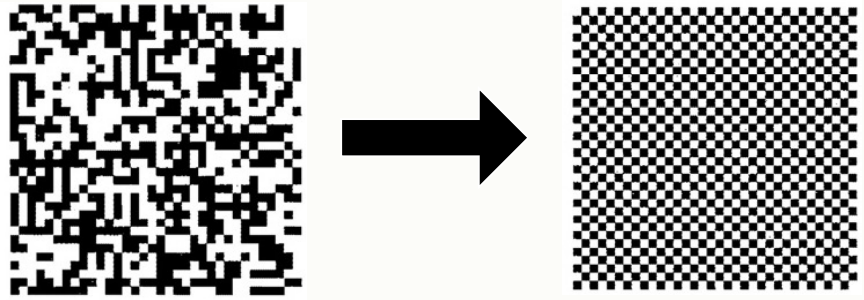


The grid changes according to **transition rules** which determine the future state of the cell based on the present state of both the cell and its **neighbourhood**.

today: checkerboard CA

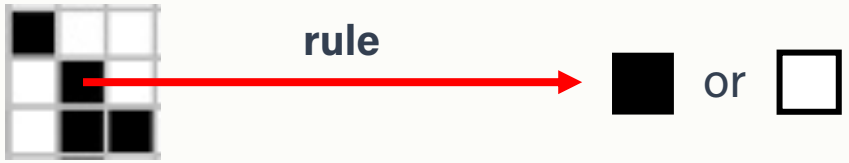
Start from an array in an initial random configuration of black and white cells

The aim is to turn the whole array into a checkerboard pattern.



Can you find such a cellular automaton?

There are $2^9 = 512$ different neighborhood-states, thus each rule is encoded by 512-digit binary string.



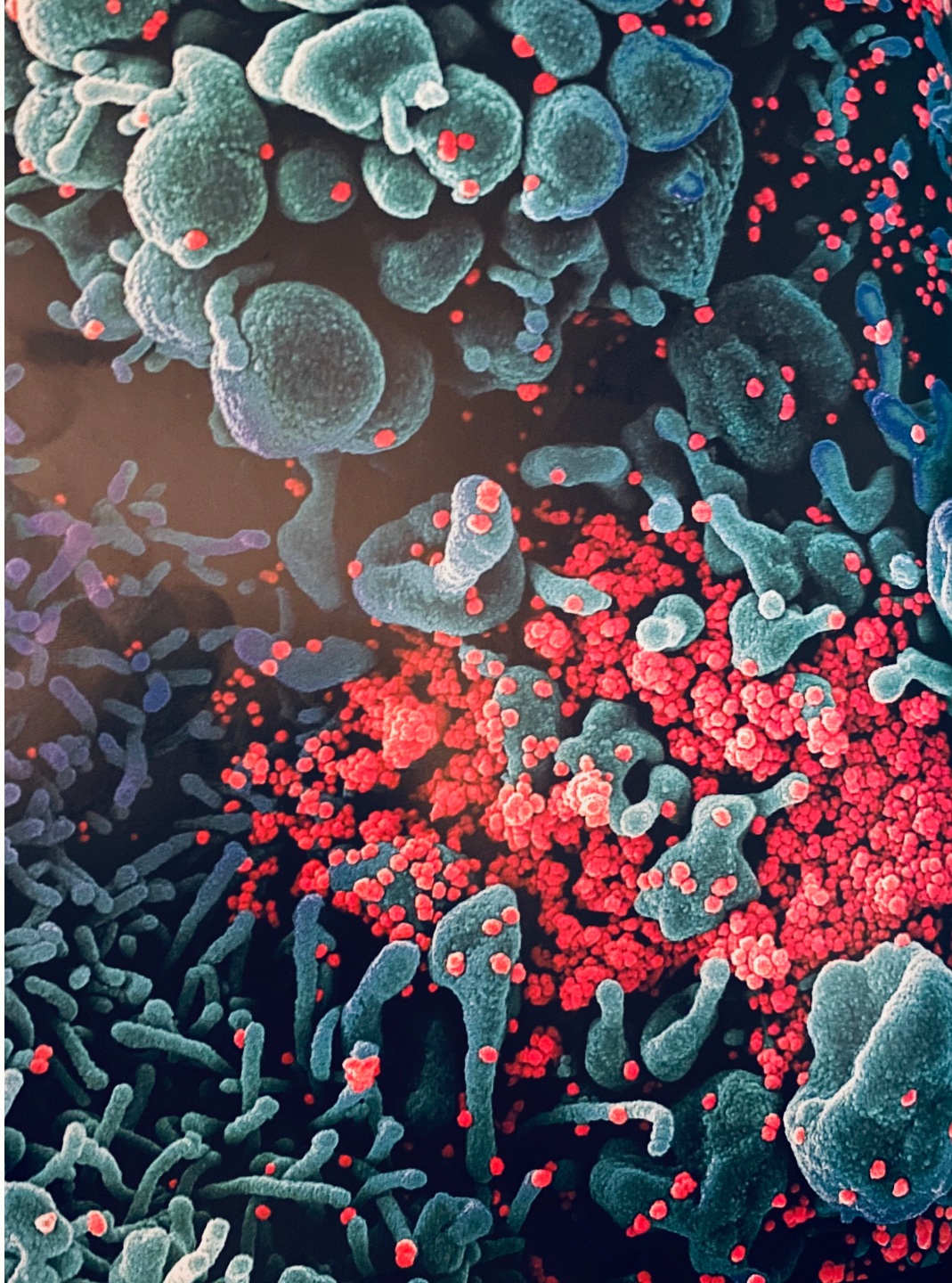
Moore neighborhood state

There are $\mathbf{2^{512}}$ automatons. How to find the one we are looking for?

$\mathbf{2^{266}}$ - number of atoms in the universe

use genetics!

- Breed cellular automata like bacteria
- Encode the rule of CA in their genome
- In each generation give them a task to perform and let the best ones survive
- Then recombine them and mutate to form a new population



chromosomes: rule encoding

Each rule is encoded by a 512-digit binary string

rule/chromosome → 1001 101 ... 110

a_0	a_1	a_2
a_3	a_4	a_5
a_6	a_7	a_8

$$N = \sum_{i=0}^{8} 2^i a_i$$

Genes: Nth place in the rule string determines the state of the center cell in the next timestep (0 or 1)

Iterations: new grid = Rule[N(old grid)]

periodic boundary conditions

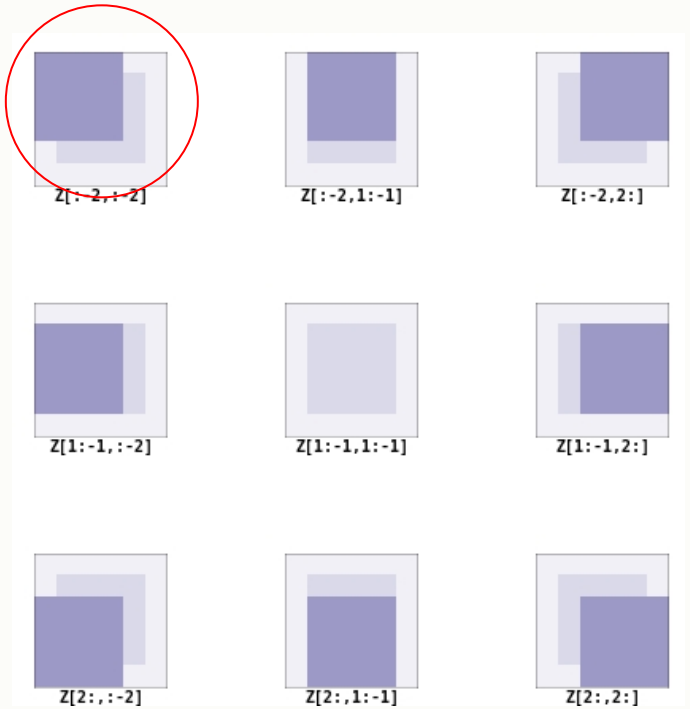
$$N = \sum_{i=0}^{8} 2^i a_i$$

a_0	a_1	a_2
a_3	a_4	a_5
a_6	a_7	a_8

To calculate N you can use numpy's **roll** function,

e.g. **np.roll(np.roll(grid, -1, axis=0), -1, axis=1)**

shifts the grid (periodically) by (-1,-1)



fitness function

Each chromosome is given a **fitness test** to assess how well it converts
an array of random black and white cells into an approximation of a checkerboard.

The function can be, for instance:

1. For a given rule, generate a random initial grid and perform 100 iterations of the rule
2. For each cell (x, y):
   A. subtract 3 points if cell (x+1, y) is in the same state as (x, y)
   B. subtract 3 points if cell (x, y+1) is in the same state as (x, y)
   C. otherwise, add 8 points if cell (x+1, y+1) is in the same state as (x, y) or substract 5 points if it is not; repeat for cell (x+1, y-1)
3. Repeat this for a few random initial grid configurations and take an average
4. Use different initial configurations for each new generation

You can experiment and try different ideas!

cloning

Clone each chromosome a number of times that depends on its level of fitness (consider using numpy's random.choice function)

$$f_i = \frac{F(c_i)}{\sum_k F(c_k)}$$

fraction of the ith chromosome in the cloned population

fitness of chromosome k

reproduction

Pick pairs of chromosomes randomly from the population and let them be "parents".
Two ideas for how babies are made:

uniform crossover

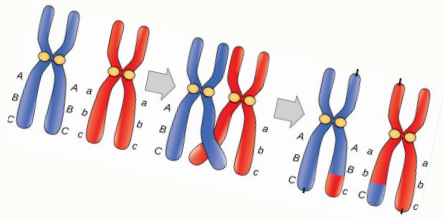
1001 101 ...110

1110 110 ...000

→ 1101 110 ...000

Align their chromosomes and create a new "baby" rule by selecting one of the state values at each of the 512 positions randomly from either one of the parents.

one-point crossover



1001 101 ...110

1110 110 ...000

→ 1001 110 ...000

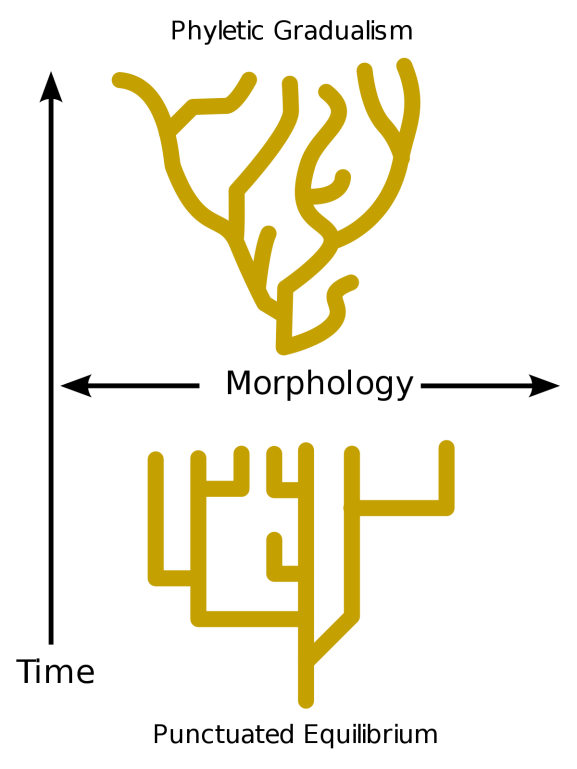
Choose a random crossover point at one of the 512 positions. Genes up to this point come from one parent and from this point onward from another parent.

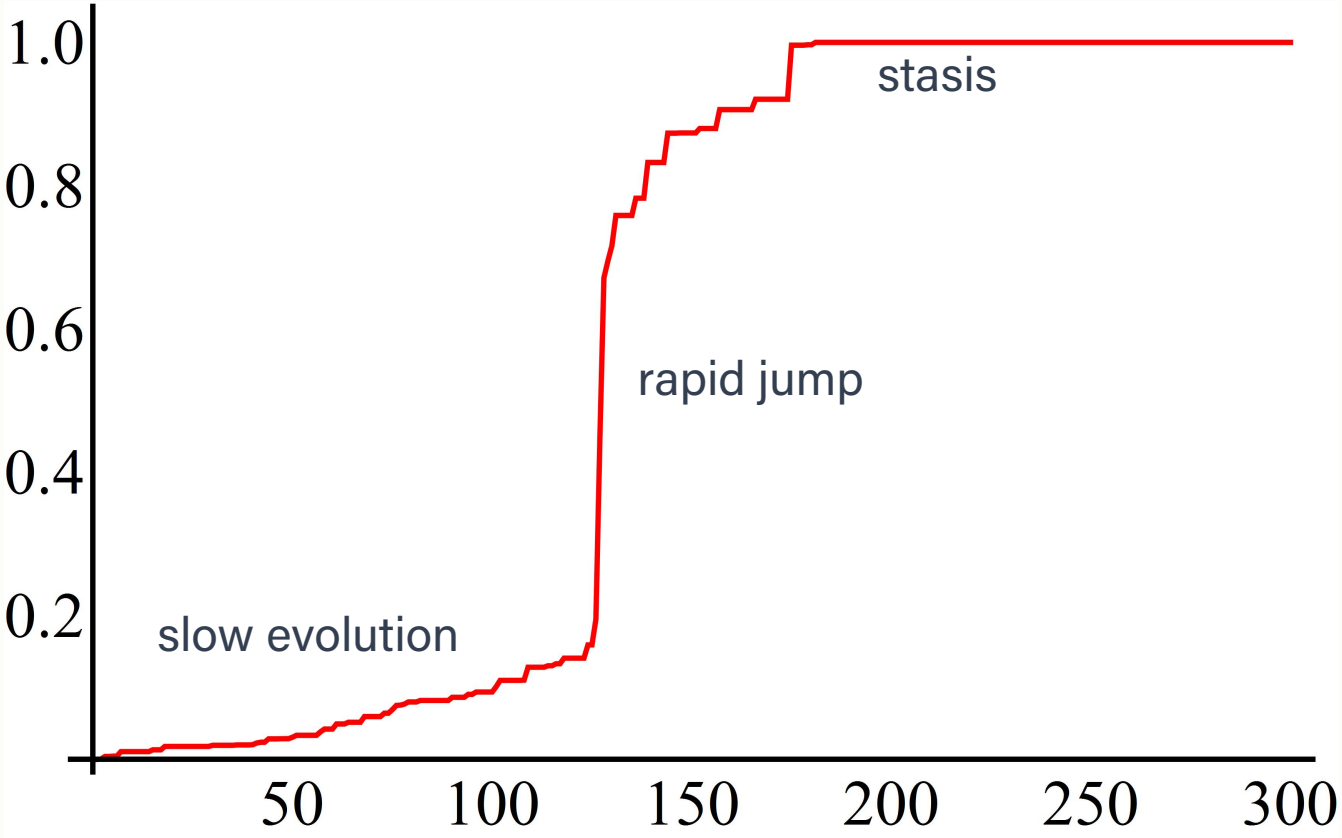
mutation

Randomly flip 1-3 bits in some of the baby chromosomes.

fitness vs time

punctuated equlibrium: significant evolutionary changes are restricted to rare and geologically rapid events followed by periods of morphological stability





species are generally morphologically stable, changing little for millions of years

this leisurely pace is "punctuated" by a rapid burst of change that results in a new species

your task:

1. try to find an effective “checkerboard CA” for grid size 50x50
2. experiment with parameter values (population size, mutation rate etc.)
3. plot a fitness function (max/avg in the population) vs time

CA TOURNAMENT (winner gets 1 extra point)

- submit your best CA to me by e-mail, give it a name
- we will make them fight next week,
  live, in a bracket tournament
- the games will be played on 500x500 grids
- both fitness and speed of the CAs will determine the winner

some hints:

- Write separate functions for **grid generation, fitness assessment**, **neighborhood value calculation**, **selection**, **reproduction**, **mutation**, etc. Test them on small examples for which you can verify the correctness yourself. Those numpy rolls and axes can get tricky.
- It is good to keep the size of the population constant – balancing cloning, reproduction and deaths.
- The evolution algorithm can be changed – for example, you can give up cloning but instead use the reproduction probability which depends on the fitness level. You can look up "selection (genetic algorithm)" on Wikipedia for other ideas on how to select.
- When assessing the fitness, it is good to average the value over the last few timesteps.
- The size of the population does not need to be large – very good results have been obtained even with the population of just 10 chromosomes, 5 of them (above the median) surviving and 5 offsprings; but there were some algorithms based on the population of 50 chromosomes also performing very well – it is all up to you.
- You can begin by breeding the cellular automata on a 50x50 grid; but if you want to take part in the competition then make sure that it also performs well on larger lattices (you can move the breeding to a larger lattice when you have obtained a strong population, performing well on smaller lattices).