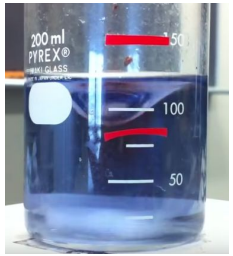
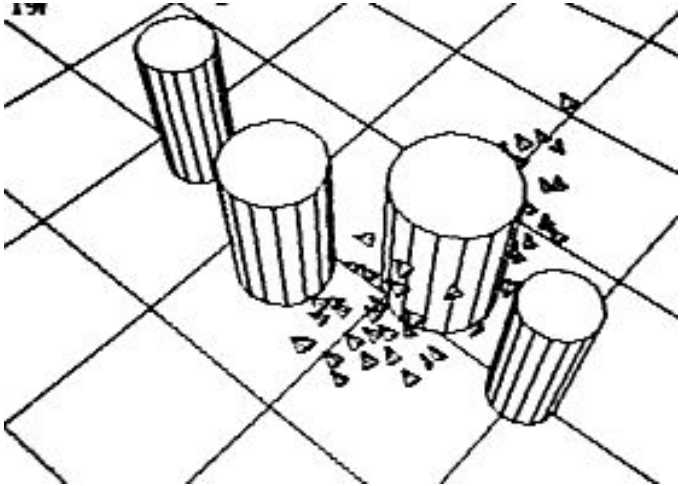
Introduction to Evolutionary Algorithms

CS369 // Computational Science

Matthew Egbert
m.egbert@auckland.ac.nz

Last lecture...



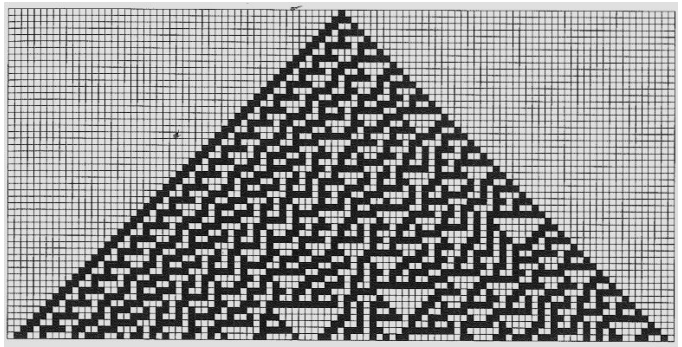


Building computational models…

… for quantitative comparison to real world systems.

… to recreate (qualitatively) an interesting natural phenomenon.

… to explore an abstraction, an idea, or set of assumptions.



Models are useful in…

… evaluating how well we understand a real-world phenomenon

… predicting real-world phenomena

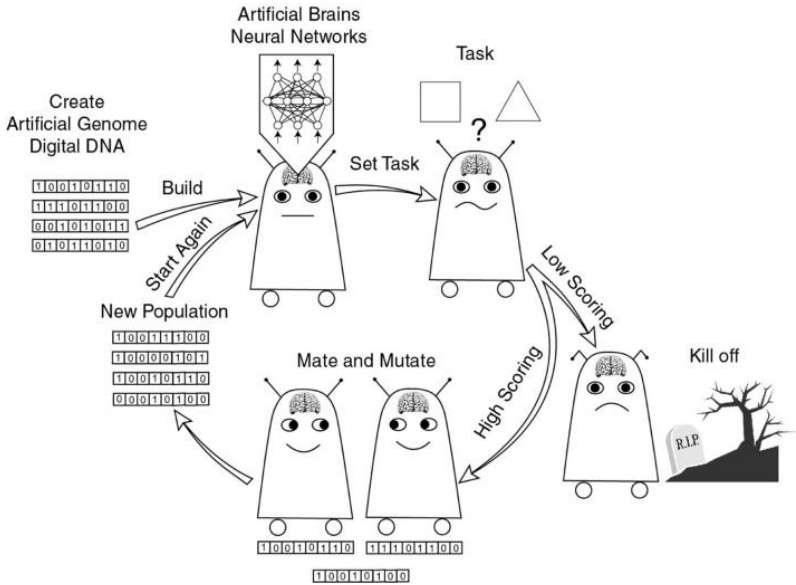
… communicating ideas.

… driving the creative process of coming up with new questions or hypotheses.

Artificial Evolution

Today I will introduce **evolutionary algorithms**, which are computer algorithms that simulate evolution allowing you to "breed" desired systems.

We'll start by considering a few reasons ***why*** scientists might use these algorithms...

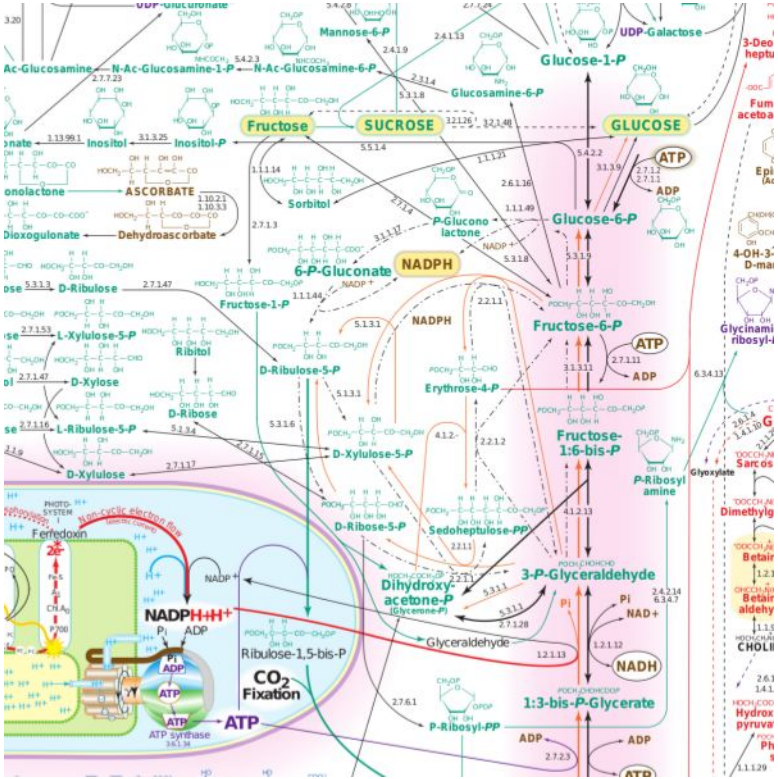


#1 Solving problems we couldn't

You are constructing a model of a metabolic network in some organism. Your data about the real world network is incomplete.

Task: select values for the free parameters so that the system behaves similarly to the natural system.

This would be hard (impossible?) for a human to do, but could be a good type of problem to use a genetic algorithm for.

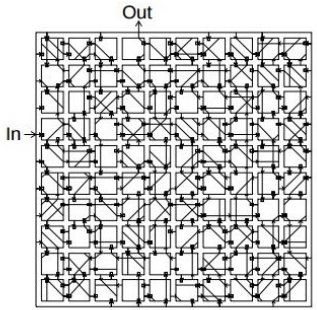


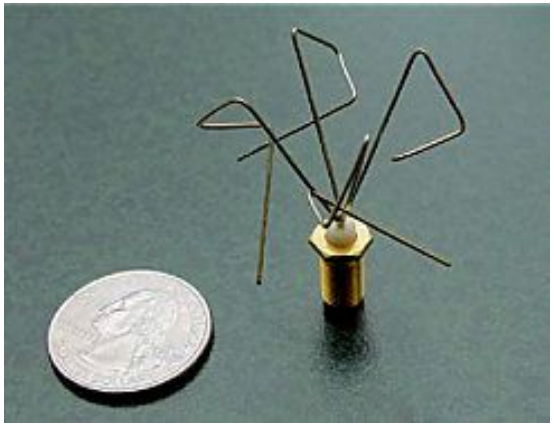
#2 Engineering artifacts we wouldn't (or couldn't?) design...

Some designs are inaccessible (or at least difficult) to by-human engineering.

Evolved FPGA Circuits

1. Tone discriminator circuit [1]
2. Oscillator (inadvertent radio) [2]





Evolved antennae.
"The resulting antenna often outperforms the best manual designs, because it has a complicated asymmetric shape that could not have been found with traditional manual design methods."

https://en.wikipedia.org/wiki/Evolved_antenna

[1] Thompson, A. (1997). An evolved circuit, intrinsic in silicon, entwined with physics. Evolvable systems: from biology to hardware, 390-405. Chicago
[2] https://www.newscientist.com/article/dn2732-radio-emerges-from-the-electronic-soup/

#3 Avoiding the bias of "by-human" engineering

Humans design understandable systems.

- Out of necessity (we kind of have to!)
- We often want to be able to predict what our creations will do.
- Accountability

Evolution has no such constraint!

Biological systems are messy! Consider, for instance, the brain, or gene-regulatory networks. These systems have wide variation of components, and interaction between components, cross talk, etc. Lack of hierarchy, etc. etc.

Making systems understandable is a **constraint.** What types of organization does it preclude?

Genetic algorithms can be used to design complex systems that are

1. (hopefully) more similar to those produced by natural evolution -- or at least different from those that we would design as humans
2. still simple enough to understand

By studying the product of artificial evolution, we hope to improve our understanding of the product of natural evolution.

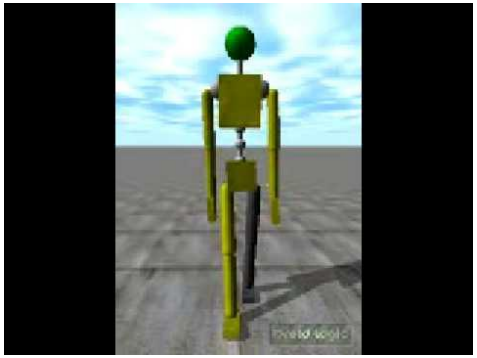
Evolutionary Robotics

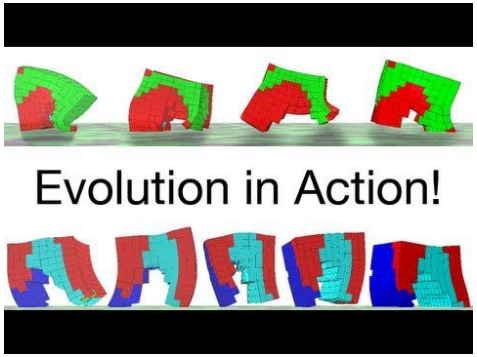
Example research areas where artificial evolution is used include...

- bipedal walking
- soft-bodied robotics
- "emergent engineering" e.g. swarming or collective behaviours

Again the idea:

- engineering biases different from "by-human engineering"
- systems that are simple enough to understand





Outline

Quick refresher on basics of evolution

An evolutionary algorithm for designing a paper airplane

Terminology: phenotype, genotype, fitness, fitness landscape, selection, mutation

Pseudocode for a GA.

Details of genotypes, methods for selection, mutation, crossover etc.

Evolution occurs when…

..there is a **population** of individuals

..each individual **dies**

..each individual can **reproduce**

..there is **heritable variation** in the population, meaning:

> **variety:** some individuals are **more fit** (i.e. more likely to reproduce) than others.
>
> **heritability:** traits that contribute to fitness are passed from one generation to the next.

Over successive generations, the population changes, "**adapting** to its environment."

A shorthand way of thinking about this is that “fitness increases,” but this is not strictly true and it glosses over some details.

Natural vs. artificial evolution

In **natural evolution** fitness is an abstraction of a wide variety of factors that contribute to how likely an organism is to reproduce.

These include phenotypic traits, environment, luck, other organisms (conspecifics or predators or prey or competitors), etc.

But in **artificial evolution** fitness is specified by a person. You specify what it is to be fit!

- breeding
- genetic algorithms

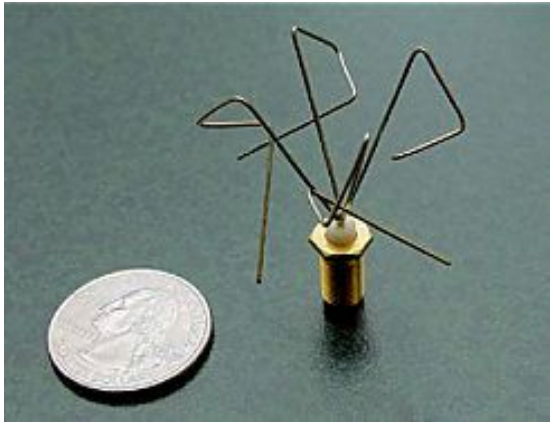
Evolutionary Algorithms

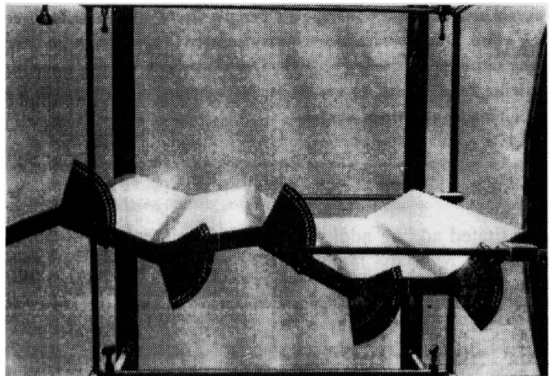
Key evolutionary dynamics (mutation, selection, reproduction, etc.) are simulated so as to evolve a desired system…

- reaction rates to make a model of metabolic dynamics match empirical data
- a class schedule that minimizes conflicts
- the shape of an antenna or airplane wing
- just about anything you can think of! (although in practice it is often not so easy)

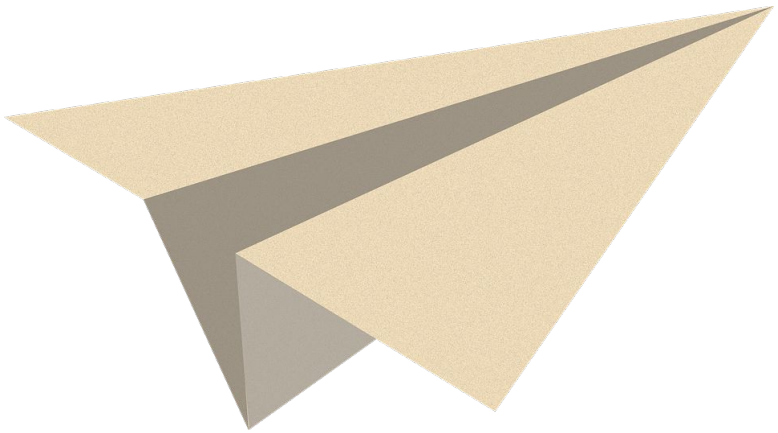
The remainder of this lecture and much of tomorrow introduces the basics of genetic algorithms.

Along the way, we briefly consider why it can be difficult for artificial evolution to find a solution, and a few methods that can be used to improve chances of success.





A genetic algorithm to design a paper airplane...



Evolving a paper airplane

1. Generate 20 random sequences of folding instructions
2. Fold each piece of paper according to instructions written on them
3. Throw them all out of the window
4. Pick up the ones that went furthest, look at the instructions
5. Produce 20 new pieces of paper, writing on each bits of sequences from parent pieces of paper, occasionally making a mistake.
6. Repeat from (2) on.

Fold TL to BR towards you
Fold horizontal middle away
Fold vertical middle towards

Fold **TR** to BL towards you
Fold horizontal middle away
Fold vertical middle **away**

Selection

1. Generate 20 random sequences of folding instructions
2. Fold each piece of paper according to instructions written on them
3. Throw them all out of the window
4. Pick up the ones that went furthest, look at the instructions
5. Produce 20 new pieces of paper, writing on each bits of sequences from parent pieces of paper, occasionally making a mistake.
6. Repeat from (2) on.

Selection

1. Generate 20 random sequences of folding instructions
2. Fold each piece of paper according to instructions written on them
3. Throw them all out of the window
4. **Pick up the ones that went furthest**, look at the instructions
5. Produce 20 new pieces of paper, writing on each bits of sequences from parent pieces of paper, occasionally making a mistake.
6. Repeat from (2) on.

Mutation

1. Generate 20 random sequences of folding instructions
2. Fold each piece of paper according to instructions written on them
3. Throw them all out of the window
4. Pick up the ones that went furthest, look at the instructions
5. Produce 20 new pieces of paper, writing on each bits of sequences from parent pieces of paper, occasionally making a mistake.
6. Repeat from (2) on.

Mutation

1. Generate 20 random sequences of folding instructions
2. Fold each piece of paper according to instructions written on them
3. Throw them all out of the window
4. Pick up the ones that went furthest, look at the instructions
5. Produce 20 new pieces of paper, writing on each bits of sequences from parent pieces of paper, **occasionally making a mistake.**
6. Repeat from (2) on.

Q: What would happen without mutation?

A: The population **converges** (variety is lost). If we assume that there is no randomness in measuring fitness, the population becomes homogenous -- all 9s.

Generation 1
5
2
4
4
1
9
3

Variation

Random mutations occur, maintaining variation in the population.

We saw in the previous slide that the population **CONVERGED.** At the end, everyone was a '9'.

Mutations counteract convergence, maintaining variety in the population (or equivalently producing novelty), which allows adaptation to continue.

Most mutations are **deleterious** (cause a decrease in fitness), but rarely, they increase fitness.

(Only) when there are mutations, can fitness increase above the maximum fitness of the initial population.

<u>with</u> mutation

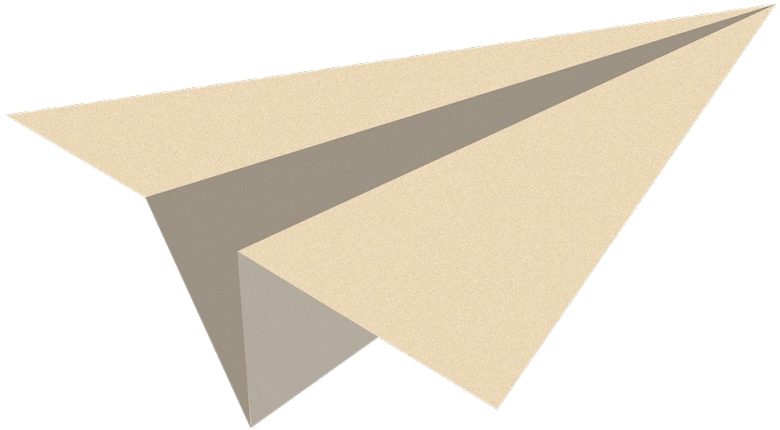
Generation 1	Generation 2	Generation 3	Generation 4
5	6	8	10
2	5	9	11
4	3	10	9
4	9	10	10
1	10	10	10
9	8	10	11
5	0	9	4

Terminology

Fitness
How good a particular solution is at solving the problem at hand.

Genotype
A specification of solution. The genotype can be **copied** and **mutated**.

Phenotype
An instantiation of the genotype. i.e. an attempt to solve a problem. A phenotype can be **evaluated** (its fitness can be measured)



Fitness

1. Generate 20 random sequences of folding instructions
2. Fold each piece of paper according to instructions written on them
3. Throw them all out of the window
4. Pick up **the ones that went furthest**, look at the instructions
5. Produce 20 new pieces of paper, writing on each bits of sequences from parent pieces of paper, occasionally making a mistake.
6. Repeat from (2) on.

Genotype & Phenotype

1. Generate 20 random sequences of **folding instructions**
2. Fold each **piece of paper** according to instructions written on them
3. Throw them all out of the window
4. Pick up the ones that went furthest, look at the instructions
5. Produce 20 new pieces of paper, writing on each bits of instruction sequences from parent pieces of paper, occasionally making a mistake.
6. Repeat from (2) on.

Remember:

the **genotype** can be copied and mutated
the **phenotype** can be evaluated for fitness
(sometimes there is essentially no difference between the two)

Alternatively...

You could be…

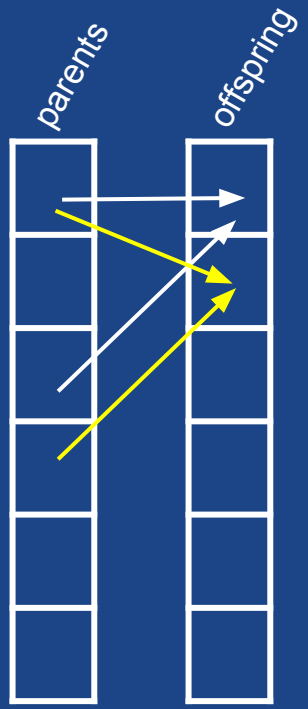
...evolving for optimal **timetables**

genotype: string encoding timetable allocations
phenotype: schedule
fitness: (negative) number of clashes

...or for optimal **aircraft wing design**

genotype: listing various wing dimensions
phenotype: actual wing (or simulation thereof..)
fitness: formula based on lift/drag/cost of wing

GA pseudocode




```
POP_SIZE = 30
N_GENES  = 10
N_GENERATIONS = 100

parents[POP_SIZE,N_GENES];
offspring[POP_SIZE,N_GENES];

## INITIALIZE THE POPULATION
for i in range(0,POP_SIZE) :
    for j in range(0,N_GENES) :
        parents[i,j] = appropriate_random_value()

## EVOLVE. Each time around the outer loop is one "generation"
for generation_i in range(0,N_GENERATIONS) :

    ## evaluate the fitness of every individual
    fitnesses = [eval_fitness(indiv) for indiv in parents]

    for offspring_i in range(0,N_GENES) :
        ## randomly pick 2 individuals from parents
        ## selecting preferentially for the fitter ones
        parent_a, parent_b = biased_random_selection_of_two_parents()

        ## recombine to make 1 offspring, and mutate the offspring
        offspring[offspring_i] = mutate( recombine(parent_a,parent_b) )

    ## overwrite the parents with the offspring, (throwing
    ## away the previous generation in the process)
    parents[:] = offspring[:]
```

Details

Genotypes

Mutation

Selection

Evaluating fitness

Genotypes

The most common genotype formats are...

A sequence of symbols from a finite alphabet. This kind of genotype is most directly comparable to the nucleotides of DNA..

...AAGAGCTTGTGAAGAGTC…

In GA's you might see binary strings..

..0101001011101011010111110111 1...

A sequence of real values.

e.g.: [0.0151, 0.4, 0.942…].

These values are often constrained to lie in a certain range, e.g. between 0 and 1, but they don't always have to be.

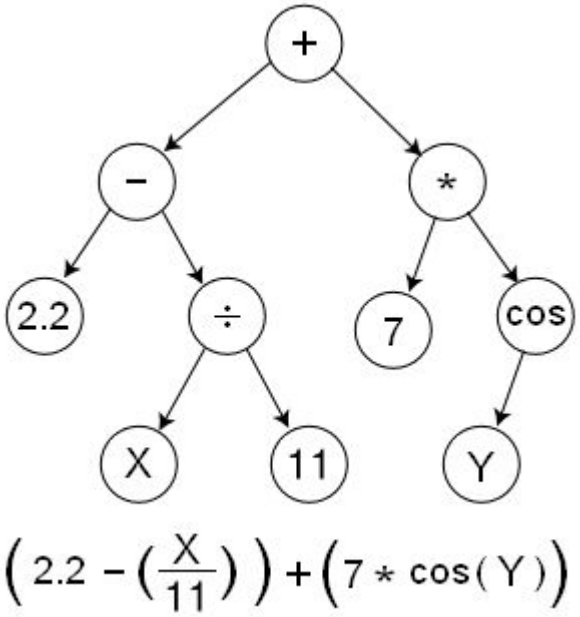
Trees etc.

There are many of alternative possibilities. Trees are also commonly used in **genetic programming** (GP), where the genotype is a **tree structure** that can be translated into a program (the phenotype).

Remember...

- genotypes (easy to copy & **mutate**)
- phenotypes (can be evaluated for fitness)

We won't spend any more time on tree-based genotypes.



Mutation

For binary strings…

For binary encodings a good (very approximate) rule of thumb is to have 1 mutation per bit. So each time you generate an offspring, each bit has a 1/N chance of being flipped where N is the number of binary genes.

For real numbers…

For real encodings the approach is often quite different form of “**creep mutation**,” where you mutate every gene by a small amount. I use a value selected from a Gaussian distribution with a standard deviation of 2% of the total allowed range of the value.

Limiting the range of genes

Mutation can cause genes to leave their allowed range. How can you best ensure that this doesn't happen?

Three possible solutions are the following (presuming an allowed gene value between 0 and 1):

Clamping 1.06 → 1

Bouncing 1.06 → 0.94

Wrapping 1.06 → 0.06

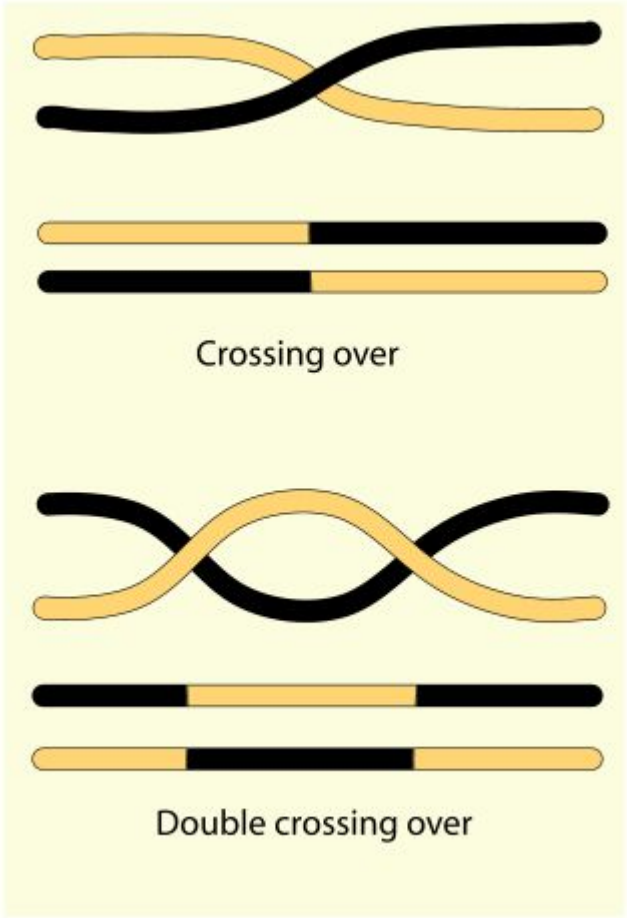
Which of these is best to use can depend upon the problem at hand, but generally bouncing and wrapping are seen as better than clamping, as clamping increases the number of mutations that produce gene values at the limits, biasing the mutation.

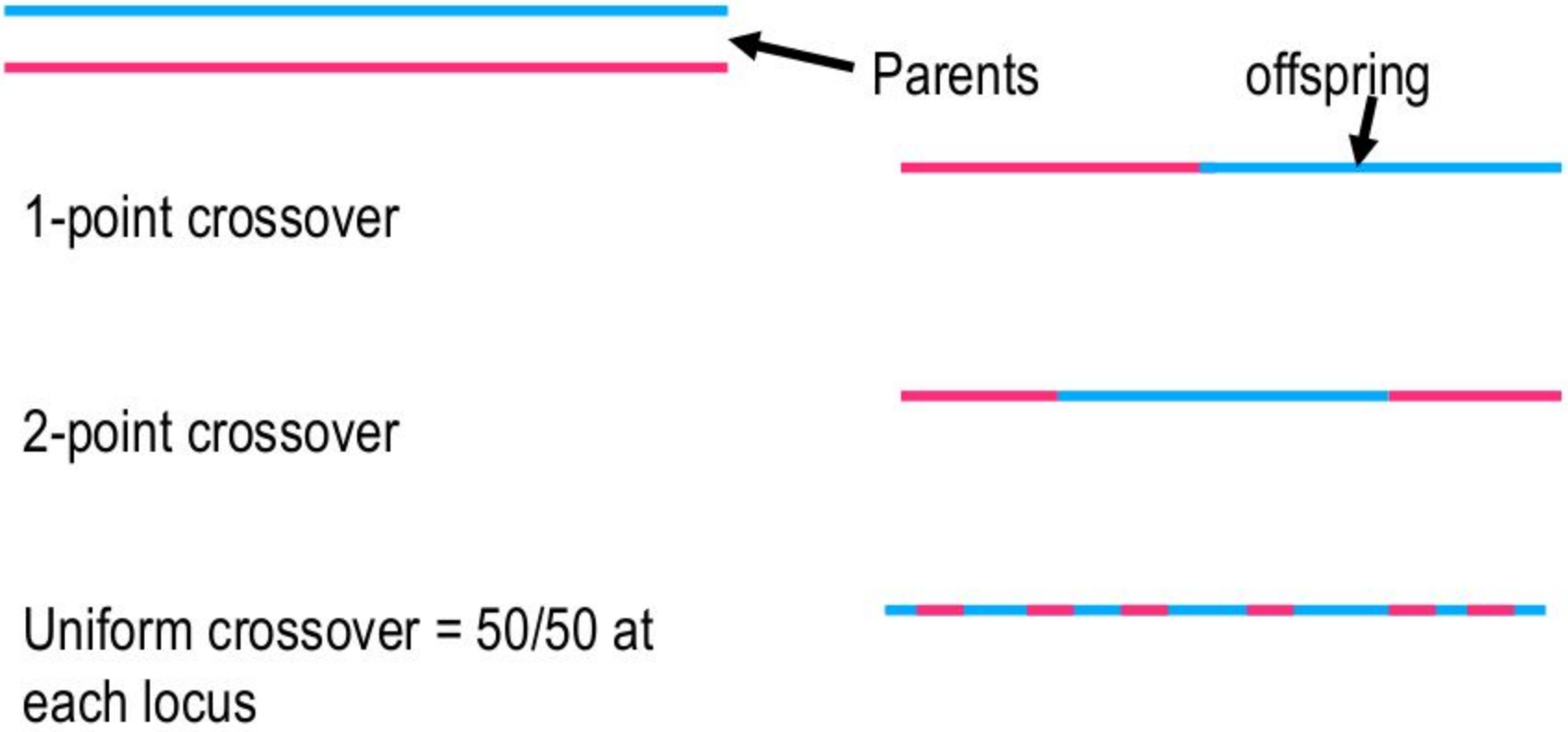
Q: Wrapping can also have negative effects, can you imagine what they might be?

Crossover

Crossover is a process that occurs in nature in which genes from parents are recombined mixed.

You don't get all of your genes from your mom OR your dad, you get a mix of each of their genes – in fact a mix of each of their chromosomes.





Cross-over for linear genotypes

Fitness landscapes

Q: What is the problem with having too low a mutation rate?

Q: What is the problem with having too high a mutation rate?

Q: When using creep mutation of real valued genes, why a mutation amount from a Gaussian distribution better than selecting from a flat distribution?

Q: Why when mutating a bitstring is it better to mutate each bit with p=1/N, rather than selecting one bit at random to flip?

Q: What exactly is the disadvantage of "wrapping" gene values to keep them within bounds?

The **fitness landscape** is a useful concept that can make it easier to think about evolutionary algorithms…

To understand the concept of a fitness landscape, let's first look at the concept of a "search space"

Searching parameter space

We can think about evolution as a **search process**.

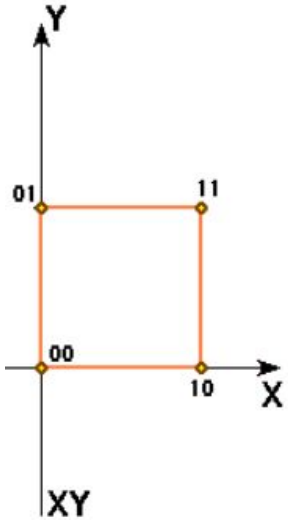
In this way of thinking evolution is searching for a set of parameters (gene values) that maximise fitness.

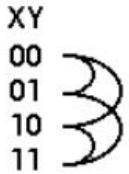
We can thus conceive of the "search space" or "parameter space" as the set of all parameters.

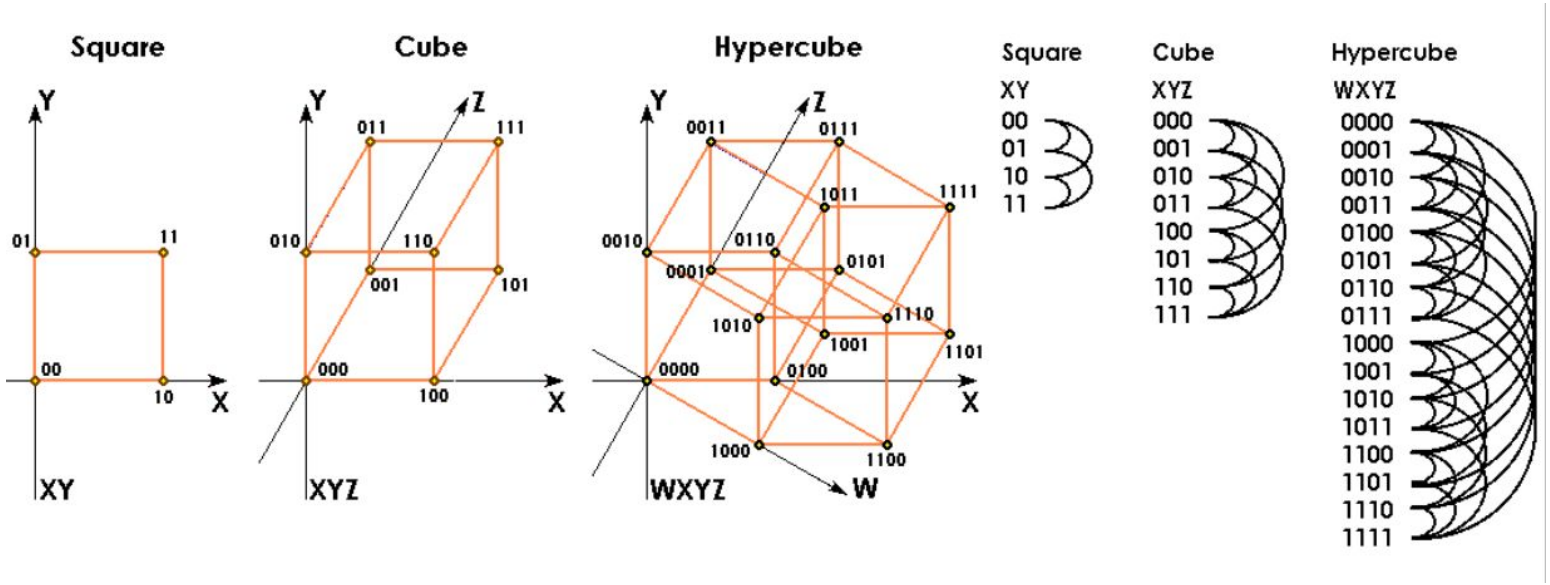
The mutation operator implies a measurement of distance in this space.

On the right is a diagram of the search space for a genotype with 2 binary genes, **X** and **Y**.

Each line represents a single mutation.





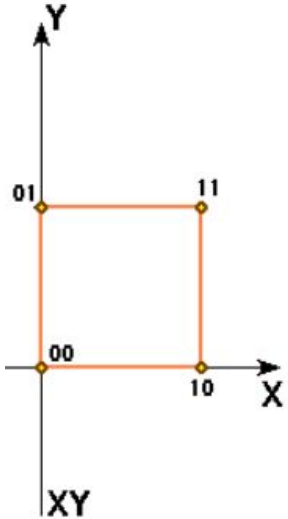

Square
Y
01
11
00
10
X
XY
Cube
Y
Z
011
111
010
110
001
101
000
100
X
XYZ
Hypercube
Y
Z
0011
0111
1011
1111
0010
0110
0001
0101
1010
1110
1001
1101
0000
0100
X
1000
1100
WXYZ
W
Square
XY
00
01
10
11
Cube
XYZ
000
001
010
011
100
101
110
111
Hypercube
WXYZ
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111

Every possible point in the search space has a fitness associated with it.

Usually we don't know what it is (if we did, we wouldn't have to run the evolutionary algorithm, we would just pick the system with highest fitness!

For a 2D search space, we can imagine the fitness as another dimension (coming off the screen), and the job of the GA is to climb the tallest hill in this **fitness landscape.**

This is actually a bit easier to see in a genotype of two continuous genes...



DEMO

Thank you!