Outline

- Overview of evolutionary computation
- Examples of evolutionary computation
- Focus on one approach: Genetic algorithms

Quick quiz

What does Crossover mean in genetic algorithms?

A. Swap a member of one population to another
B. Combine the representations of two individuals
C. Take a step of a different algorithm (e.g. Hill climbing)
D. Have an individual “die” (remove from population)

Which algorithm we've learned about previously also uses a population?

A. Simulated annealing
B. A Star
C. MCTS
D. Beam search

True or false: Crossover is applied to every member of a population

A. True
B. False

Population-based methods

Operate on a population of of initially random candidate solutions

- Beam Search
  - We used this to find solutions to TSP
- Genetic Algorithms
  - Originally designed to operate on sequences of bits
  - Also has been used on sequences of ints and floats
- Genetic Programming
  - Evolves programs
- This general field is known as Evolutionary Computation

Population-based methods are inspired by biological evolution

- Evolution is a massively parallel search method working on millions of species at a time
- Yet the rules of evolution are simple
  - Species evolve by means of random variation
  - Followed by natural selection
  - The fittest survive to reproduce
- Biological organisms are robust and adaptive to their environment in ways that we would like artificial agents to be

DNA Inspiration

Evolution ”finds” the creatures best adapted to their environment

- Better adapted individuals are more likely to survive and reproduce
- Each individual is defined by it’s genetic sequence *DNA (Strings of A, C, G, T)*

TGTTCAAGTATC



AGGGATCTGGATT



GTCCTACTAATC



CGCTACACTCAC



ATCGCATCAACC



ATCGCATCAACC



CGCAGCCAGTTA



DNA Inspiration

A population of individuals will be evaluated by their environment

- E.g. Faster individuals are more likely to survive and reproduce

TGTTCAAGTATC



AGGGATCTGGATT



GTCCTACTAATC



CGCTACACTCAC



ATCGCATCAACC



ATCGCATCAACC



CGCAGCCAGTTA



(Natural) Selection

A **fitness function** measures the relative fitness of individuals

- E.g. Fitness is 2 for skates or wheels, 1 otherwise



2 1 1 2 2 1 2

Parent 1 **Parent 2**

Next generation will be children of the current population

- Sample parents in proportion to fitness (total fitness = 11)
- Probability of being selected as parent: 1 / 11 or 2 /11

Crossover

Each pair of parents will combine their DNA in each of their children

- Split each parent's DNA a some point(s), combine the pieces to make children

Parents

ATCGCA | TCAACC



CGCAGC | CAGTTA



Children

ATCGCA | CAGTTA



CGCAGC | TCAACC



Mutation

In some cases DNA mutates at random, creating new properties

ATCGCACAGTT**A**



→

ATCGCACAGTT**G**



Evolution

- Selection, crossover and mutation recur over many generations.
- Eventually we converge on highly capable individuals!
- This is the inspiration for genetic algorithms

Features of genetic algorithms

- Maintain a population of candidate solutions
- Each candidate solution is evaluated using an objective function (also called a Fitness function) that returns a score
- Operators that may be used
  - Selection: Fitter individuals are more likely to be chosen to reproduce (like survival of the fittest)
  - Crossover: A way to combine two good candidate solutions to try to make better ones (like reproduction)
  - Mutation: Create random variation (like random mistakes made when genetic material is copied)

To use a genetic algorithm, we need to define these 4 functions for our problem

Pseudocode for Genetic Algorithm

```
GeneticAlgorithm(n, m, pCross, pMutate, maxGeneration)
  create random population of n bit strings each of length m
  generation = 0, bestIndividual = None, bestFitness = 0
  while (generation < maxGeneration)
    calculate the fitness of each individual in the population
    if better found, update the bestIndividual, bestFitness
    repeat until n offspring have been created
      select two members of population based on fitness
      with probability pCross, perform crossover
      with probability pMutate, mutate each bit of the offspring
    replace the current population with the new population
    generation += 1
  return bestIndividual
```

Genetic Algorithms vs. Local Search

- Like local search try to find a state with a good objective value
  - In this case the objective is represented by the **fitness function**
- Mechanically similar to stochastic beam search
  - Maintain a population of candidate solutions
  - Pick a successor population with some randomness
- Important differences
  - Successors can be combinations of individuals from the population
  - Successors can also introduce random changes to individuals in the population
  - We select survivors *before* generating successors

These changes can help us search larger more, complex state-spaces

- E.g. the parameters of a neural network!

Population-based methods

- Beam search
  - Uses selection only
- Genetic algorithms
  - Uses selection, crossover, and mutation
  - First evolutionary method to introduce crossover
- Genetic Programming
  - How could you select, mate, and mutate programs?

Simple GA example: Sum of bits

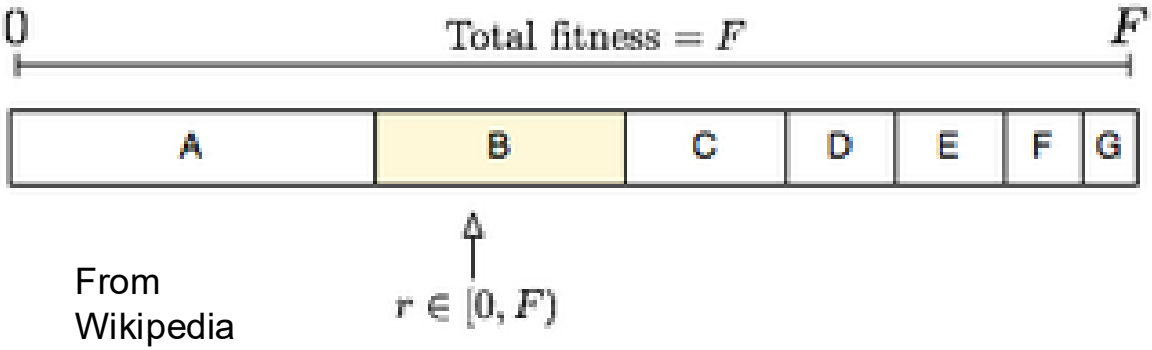
- Parameters:
  - Population size n = 4
  - Chromosome length m = 8
  - pCross=0.7
  - pMutate=0.001
- Represent individuals as lists of 1’s and 0’s
- fitness(individual) = sum of the bits

Generation 1	Chromosome	Fitness
E	11110100	5
F	00101110	4
B	11101110	6
G	00000111	3

Fitness proportionate selection

At each step randomly select parents proportional to the percentage of total fitness

- Parents are chosen in pairs to produce offspring with crossover
- Parents are chosen *with replacement*, so the same parent can be picked multiple times



From Wikipedia

Generation 1	Chromosome	Fitness	Ideal selection rate
E	11110100	5	5/18 (27.7%)
F	00101110	4	4/18 (22.2%)
B	11101110	6	6/18 (33.3%)
G	00000111	3	3/18 (16.6%)

Other possible selection strategies

Elitist selection

- Select the only the most fit members of each generations

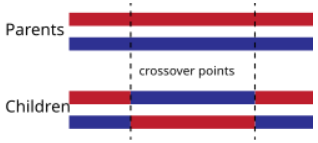
Cutoff selection

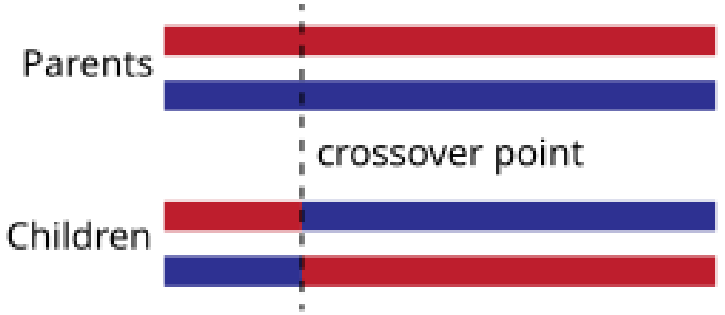
- Require a minimum fitness for each selected individual

One-point crossover

Select a random point in our bit string, create 2 children by swapping the second part of each parent

- Multi-point crossover is also possible





From Wikipedia

Select two parents from the population based on fitness, such as `B:11101110` and `D:00110100`

- Perform crossover at a randomly selected location to create two children (E and F) with no mutation

`B:11|101110` `E:11|110100`

`D:00|110100` `F:00|101110`

Generation 1	Chromosome	Fitness	Ideal selection rate
E	`11110100`	5	`5/18` (27.7%)
F	`00101110`	4	`4/18` (22.2%)
B	`11101110`	6	`6/18` (33.3%)
G	`00000111`	3	`3/18` (16.6%)

Bitwise mutation

Flip each bit with some (small) probability

Select two parents from the population based on fitness, such as B:11101110 and A:00000110

- Suppose that no crossover occurs, so B is copied into the next population, but A is mutated into G:00000111

Generation 1	Chromosome	Fitness	Ideal selection rate
E	11110100	5	5/18 (27.7%)
F	00101110	4	4/18 (22.2%)
B	11101110	6	6/18 (33.3%)
G	00000111	3	3/18 (16.6%)

Next population

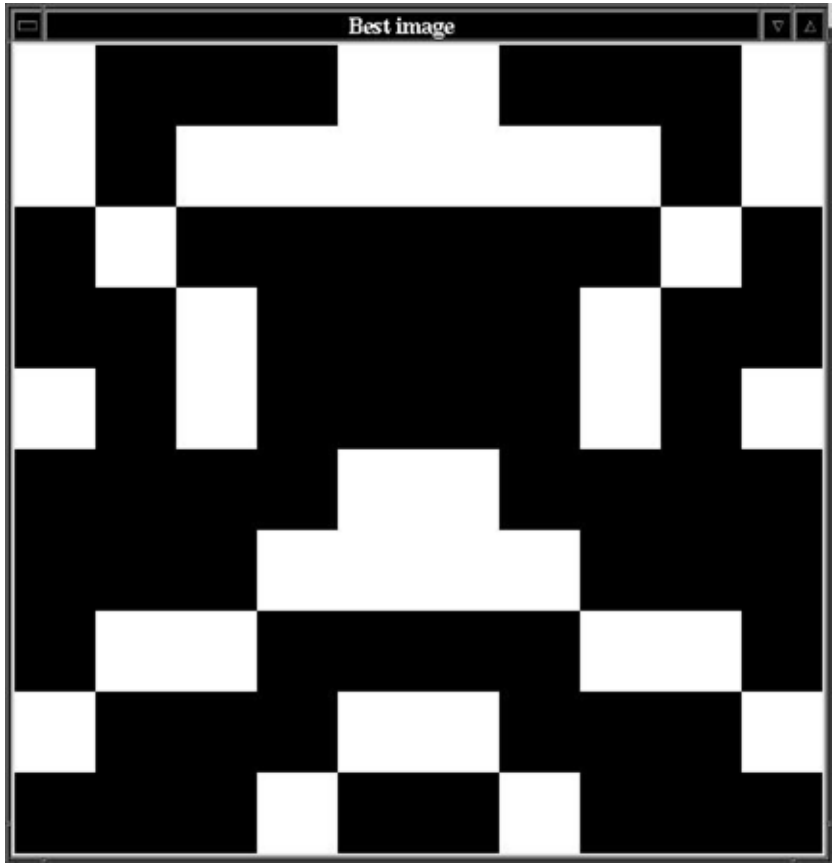
- Total fitness=18, Best=6, Average=4.5
- Typically the avg fitness increases each generation
- And better and better solutions are found over time

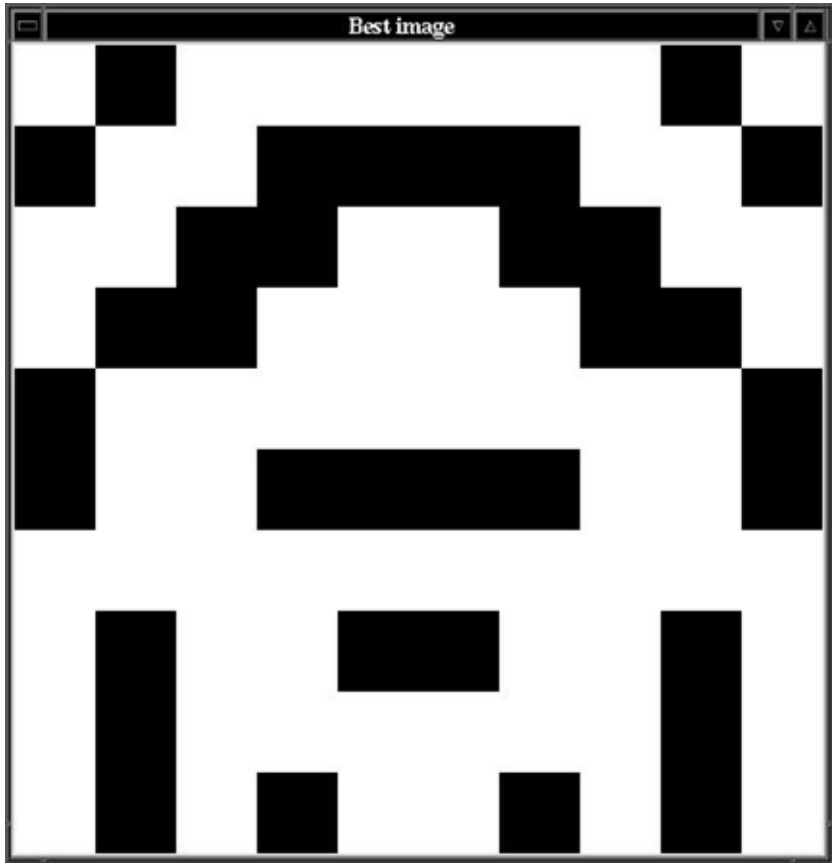
Generation 1	Chromosome	Fitness	Ideal selection rate
E	11110100	5	5/18 (27.7%)
F	00101110	4	4/18 (22.2%)
B	11101110	6	6/18 (33.3%)
G	00000111	3	3/18 (16.6%)

Example of a GA using bit strings

- Evolve black and white 2D images
- Individuals are lists of 1’s and 0’s of length 100
- Represent a 10x10 grid
- 1 = White, 0 = Black
- Fitness function: reward symmetry by giving higher scores when opposite bits in the same row are the same

Evolved images





Effects of the GA operators

- Selection gives preference to fitter individuals
- Crossover can create better individuals by recombining portions of good individuals
- Mutation maintains diversity and can introduce new features

Let’s consider how each of these works independently and then how they might interact to create a very effective search

Traveling Salesperson problem

TSP as local search

1. State representation:

Each state is an ordering of cities to visit

A -> C -> D -> B -> E -> A

0 1 2 3 4 5

2. Objective function:

The total travel cost for that order

5 + 6 + 9 + 8 + 6 = 34

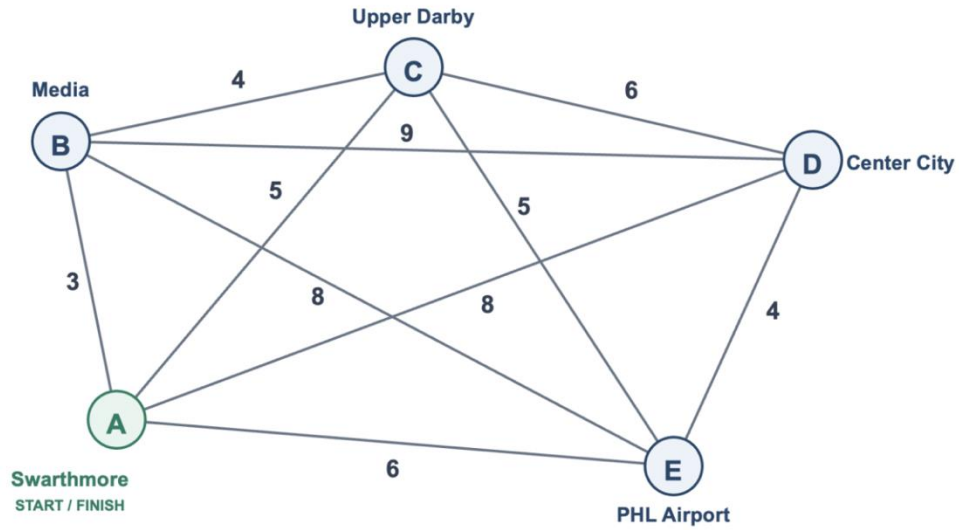
AC CD DB BE EA

3. Successor function:

Move a city to a new position in the order

A -> **E** -> C -> D -> B -> E -> A

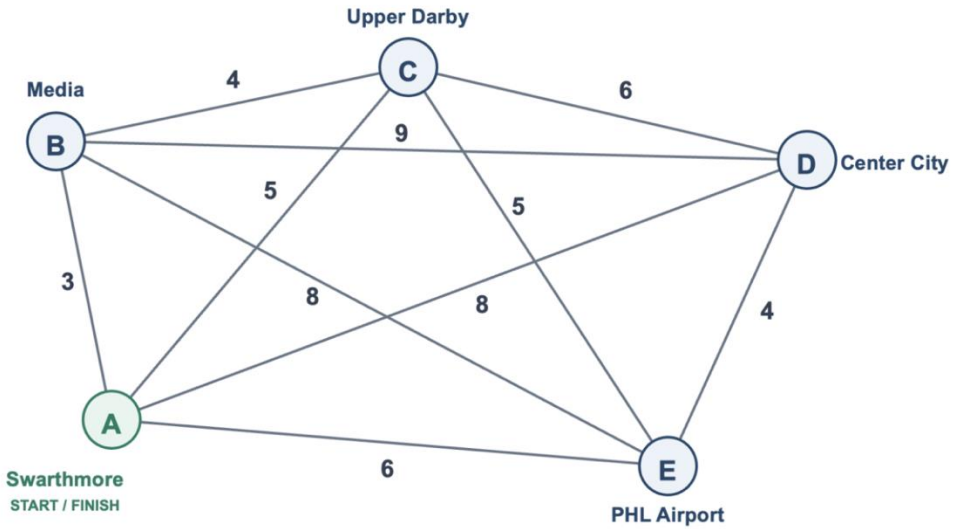
Move E to 1



Traveling Salesperson problem

How might we define:

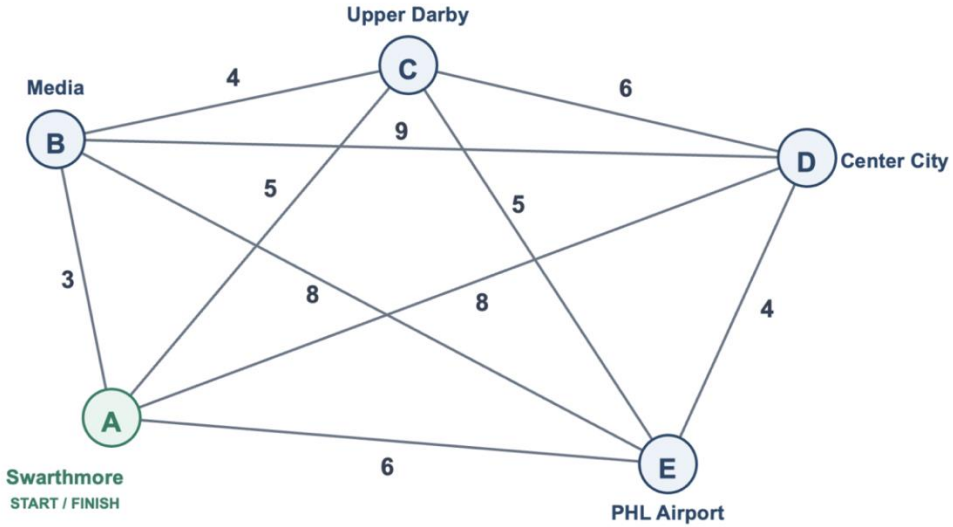
- Fitness
- Selection
- Crossover
- Mutation



Traveling Salesperson problem

How might we define:

- Fitness — 1 / Distance travelled
- Selection — Fitness-proportionate
- Crossover — Copy subset of order
- Mutation — Move city to new position

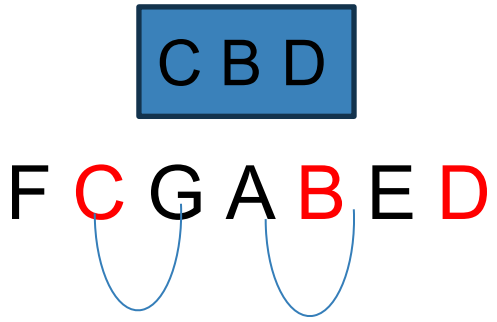


Crossover for TSP

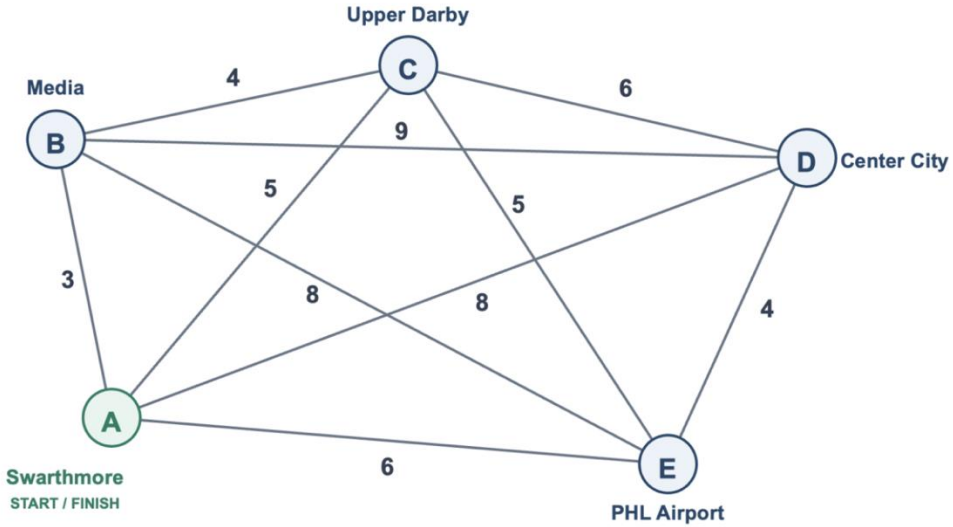
Parent 1: A E C B D F G

Parent 2: F C G A B E D

Copy subsequence from parent 1, fill remaining order with order from parent 2

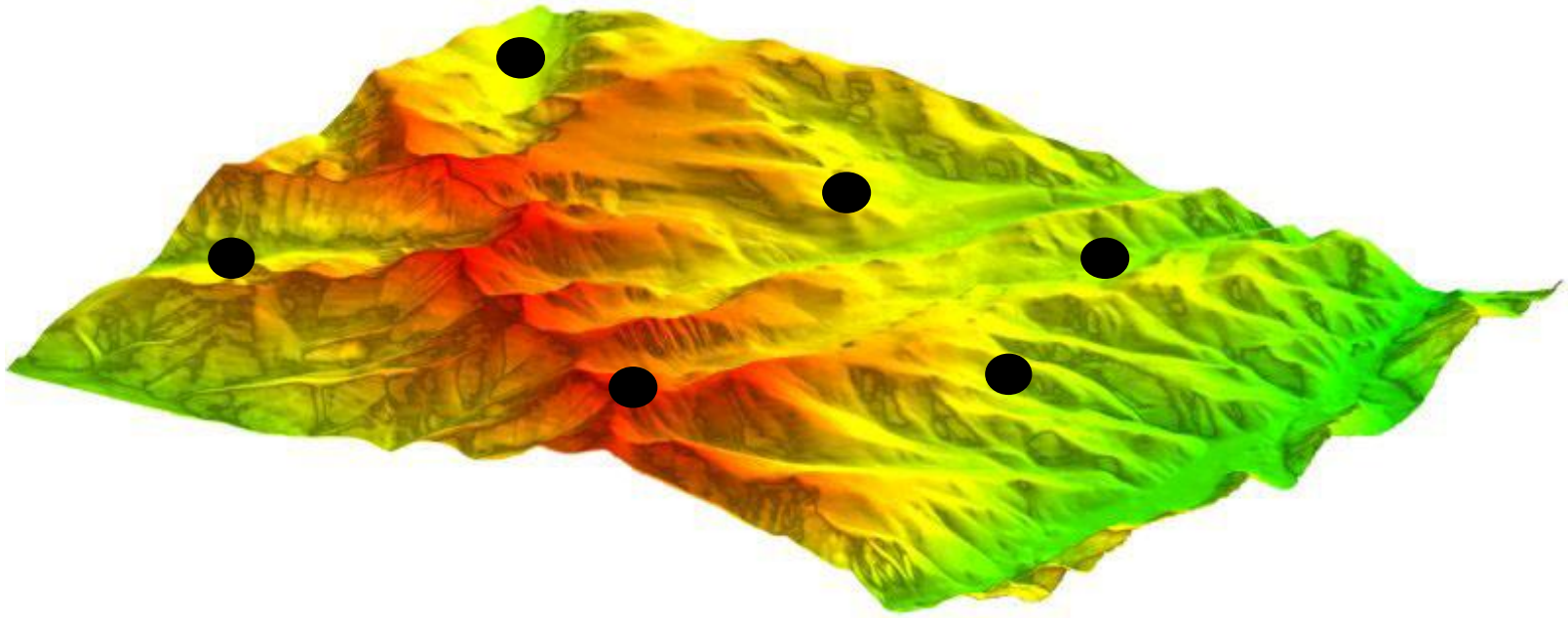


Child 1: F G C B D A E



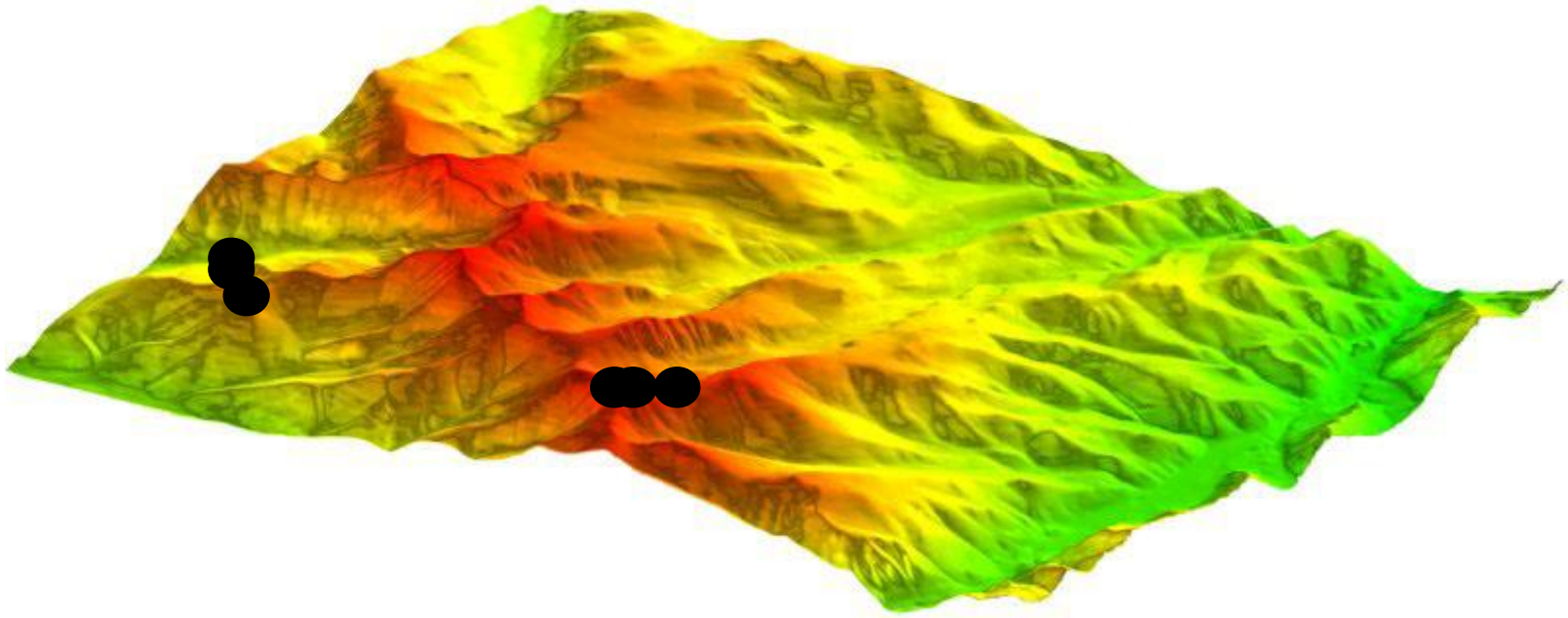
Fitness landscape where peaks represent high fitness

Initially individuals are scattered randomly across this landscape



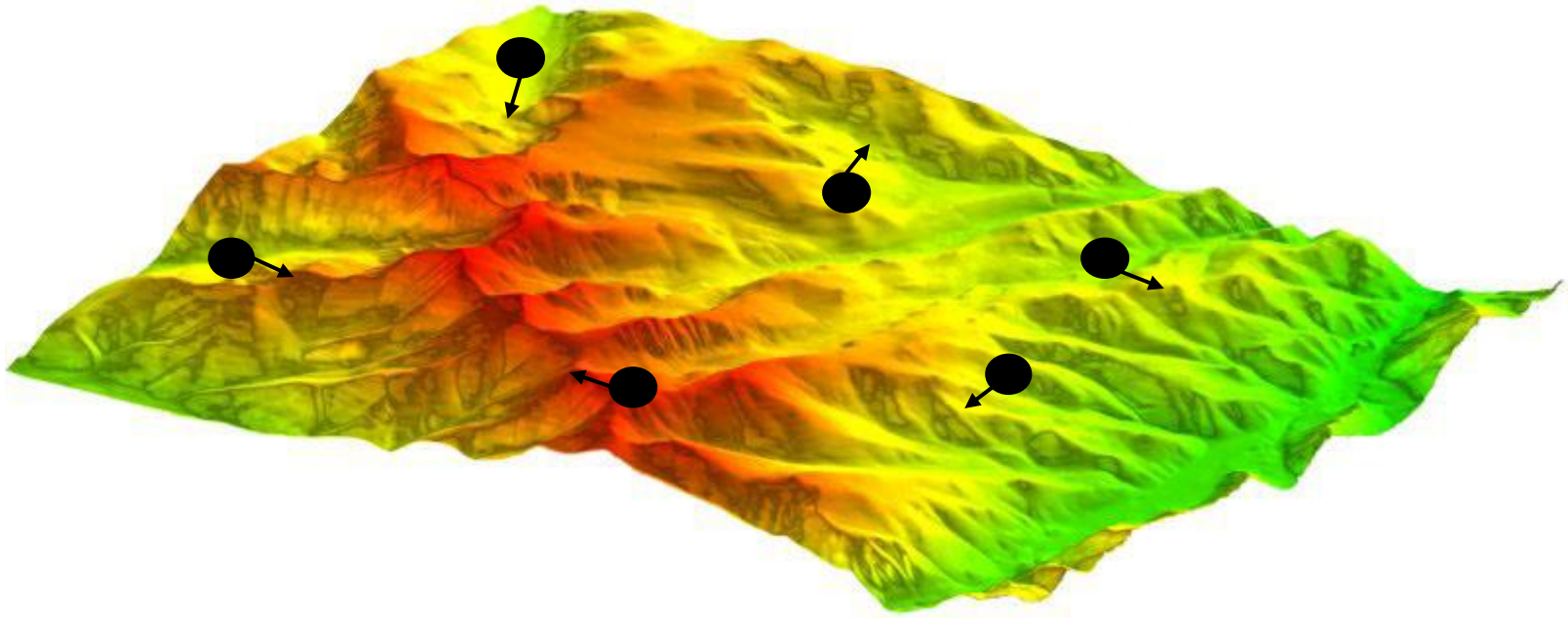
Effects of Selection alone

Would create many copies of the current best individuals (like Beam Search)



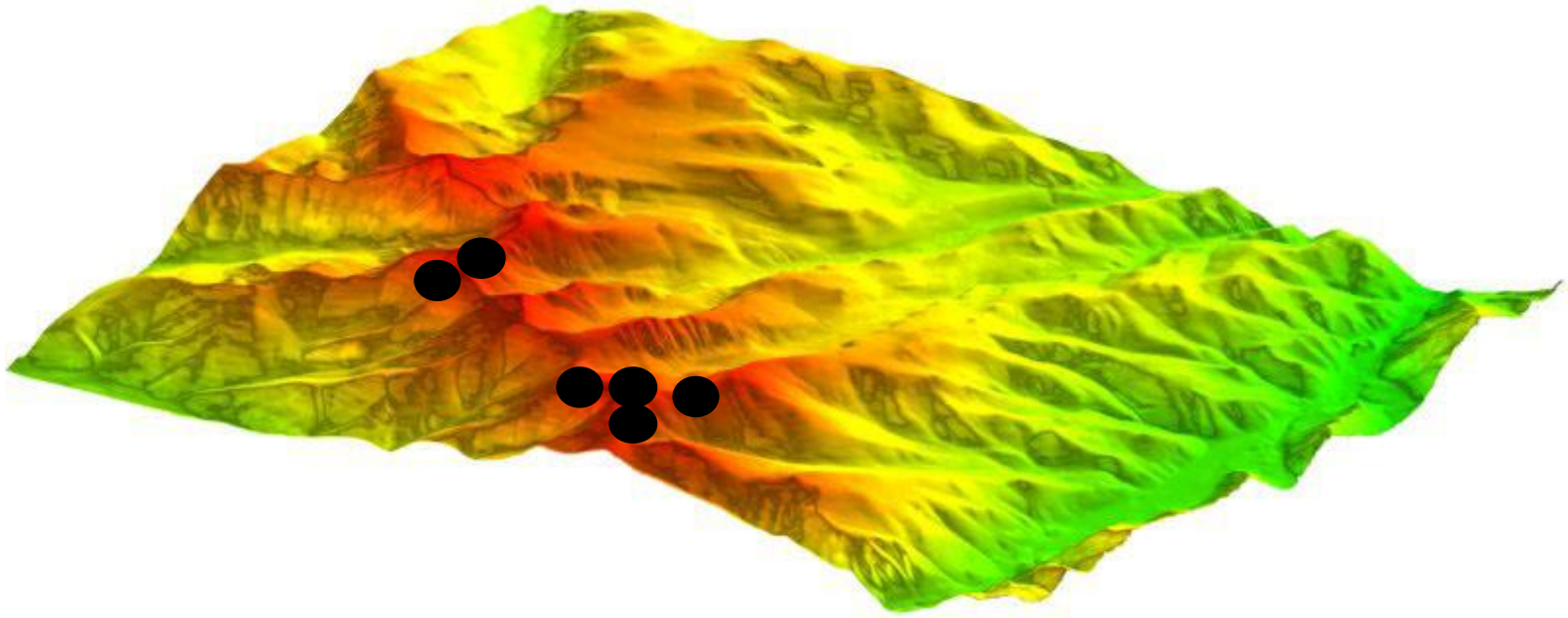
Effects of Mutation alone

Would create a random walk from the initial starting points



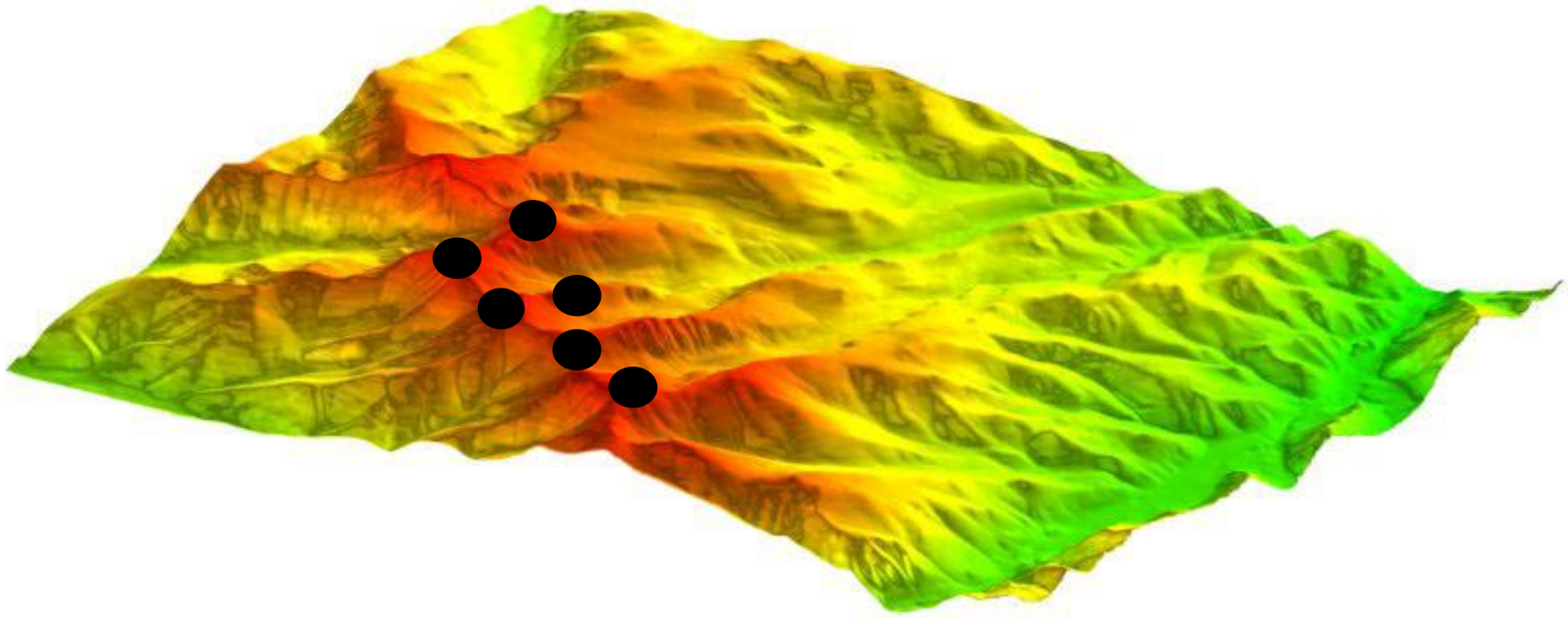
Effects of Selection and Crossover

Without mutation, will tend to converge on sub-optimal solutions



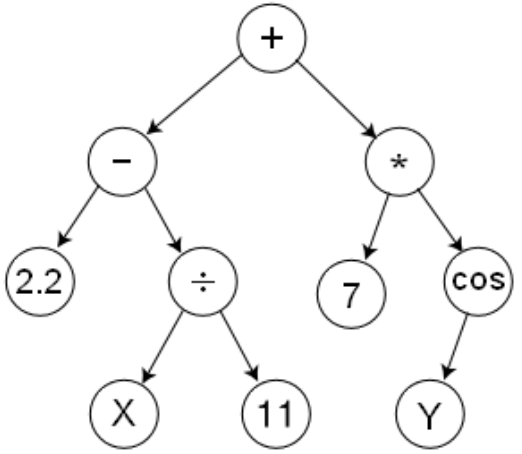
Combined effect of all three operators

Produces a distributed hill climber that is robust and can find and explore multiple peaks within the fitness landscape

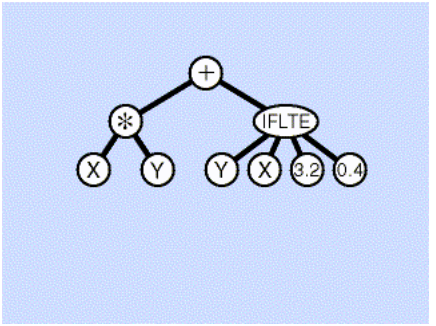


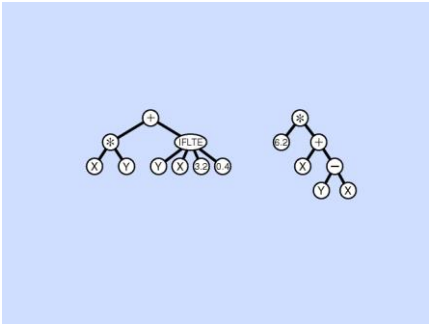
Genetic programming (GP)

- Candidate solutions are complete computer programs (originally in Lisp)
- Mutation can change operators (`+` to `−`) or functions (`sin` to `cos`)
- Mutation can also change operands
- Crossover swaps expressions within the code
- The final programs are often quite unreadable, but do the right thing (based on the fitness function)



$$\left(2.2 - \left(\frac{X}{11}\right)\right) + \left(7 * \cos(Y)\right)$$





Images from Wikipedia

Genetic Algorithms for Neural Networks?

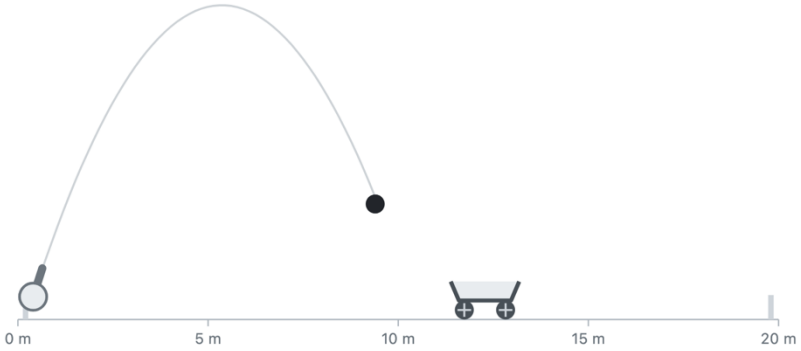
Goal: Learn the weights of the network for our ball catching game

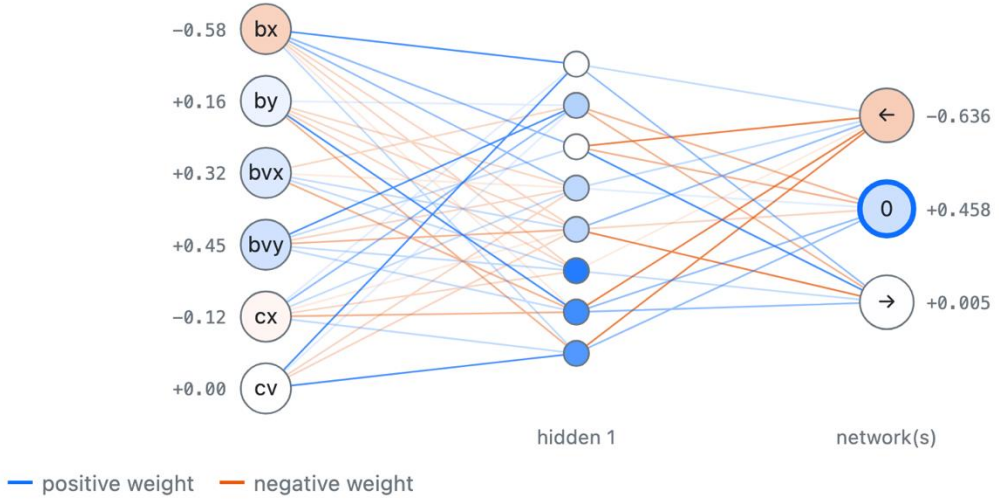
What would our representation look like?

What is our **fitness function?**

What is **crossover**?

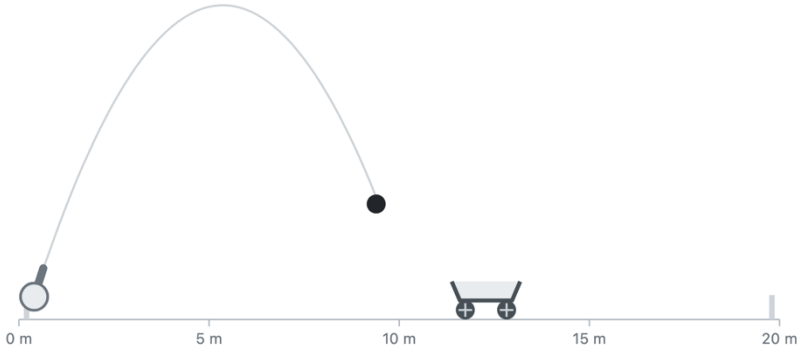
What is **mutation?**





Genetic Algorithms for Neural Networks?

Goal: Learn the weights of the network for our ball catching game



What would our representation look like?

List of all the weights!

What is our **fitness function?**

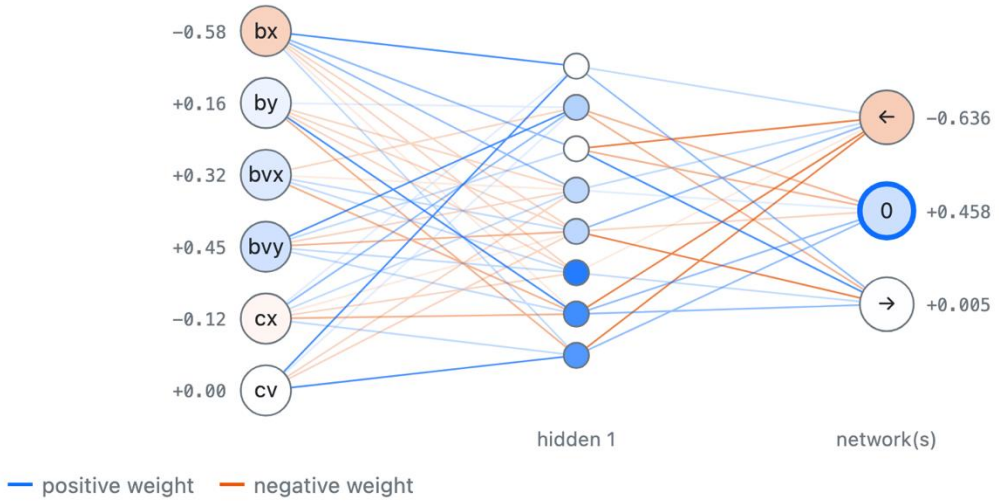
Percentage of successes (we caught the ball) over K trials

What is **crossover**?

One point crossover still works!

What is **mutation?**

Randomly change a weight…



Mutation for Neural Networks

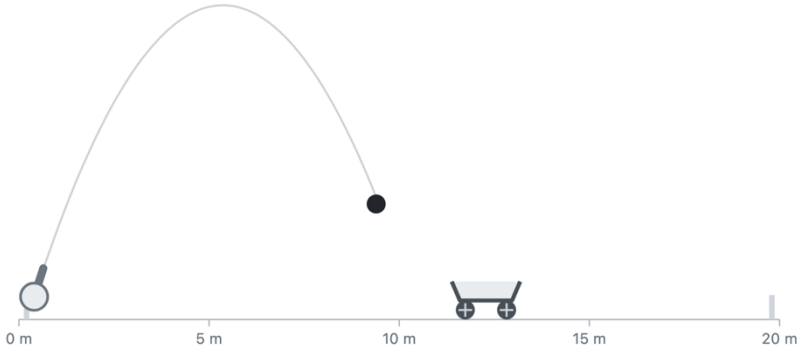
Simple mutation: Add random value to weight

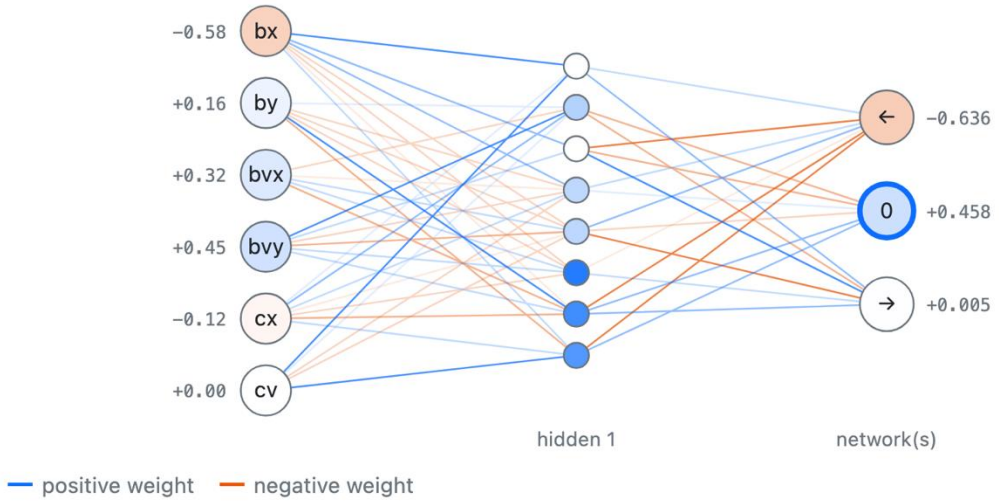
- Specify a magnitude, so we'll add a random number in the range: [-magnitude, magnitude]

$$W_i \leftarrow W_i + random(-m, m)$$

In lab code:

```
def randomWeight(self, magnitude=0.15):
    """
    Generate a random weight that has an absolute value less than
    the given magnitude, and is equally likely to be positive or
    negative.
    """
    val = random() * magnitude
    if random() < 0.5:
        val *= -1
    return val
```





Examples of Evolutionary Computation

- Fixed body, evolve the controller: Upright walking
  http://rednuht.org/genetic_walkers/
- Evolve both the body and the controller: Two-wheeled vehicles
  http://rednuht.org/genetic_cars_2/

Karl Sims: Evolving virtual creatures (SIGGRAPH 1994)

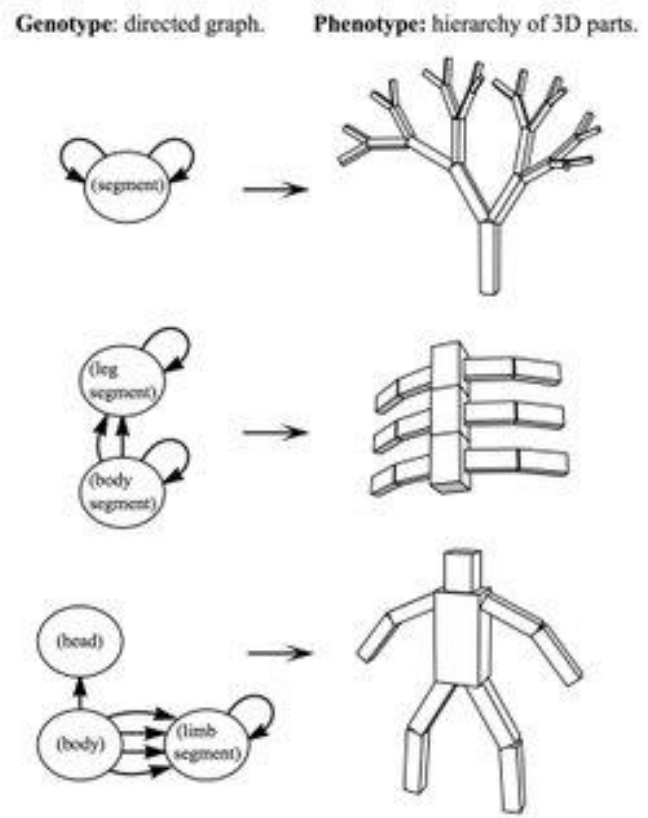


Figure 1: Designed examples of genotype graphs and corresponding creature morphologies.

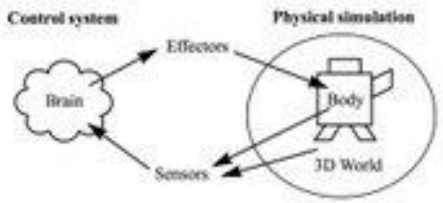


Figure 2: The cycle of effects between brain, body and world.

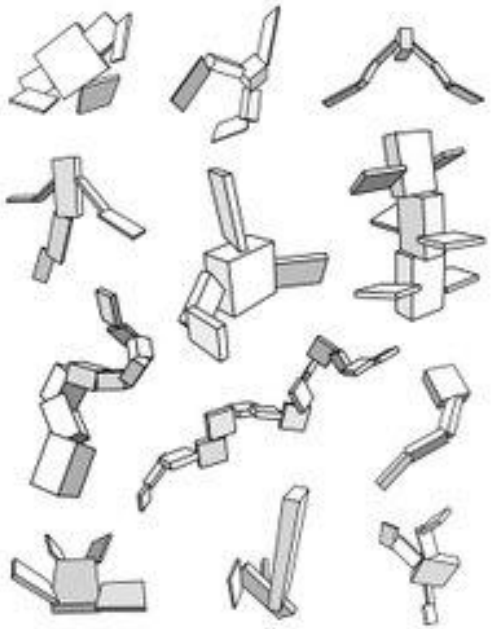


Figure 6: Creatures evolved for swimming.

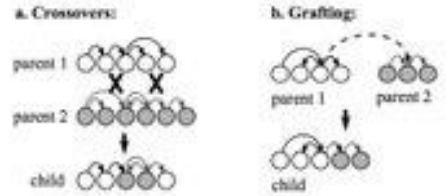


Figure 5: Two methods for mating directed graphs.

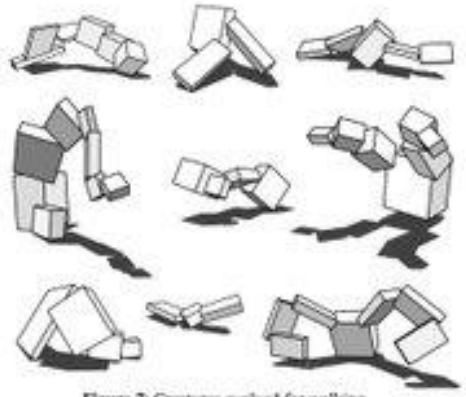


Figure 7: Creatures evolved for walking.



Figure 8: Creatures evolved for jumping.

Karl Sims: Evolving virtual creatures

https://www.youtube.com/watch?v=JBgG_VSP7f8

- Candidate solutions are represented as directed graphs of nodes and links
- Body is described as a hierarchy of 3d parts
- Control is distributed throughout the body and is nested within each body part
- Used various fitness functions in simulation
  - Swimming distance
  - Walking distance
  - Highest jump
  - Battle over a cube: Closeness to cube