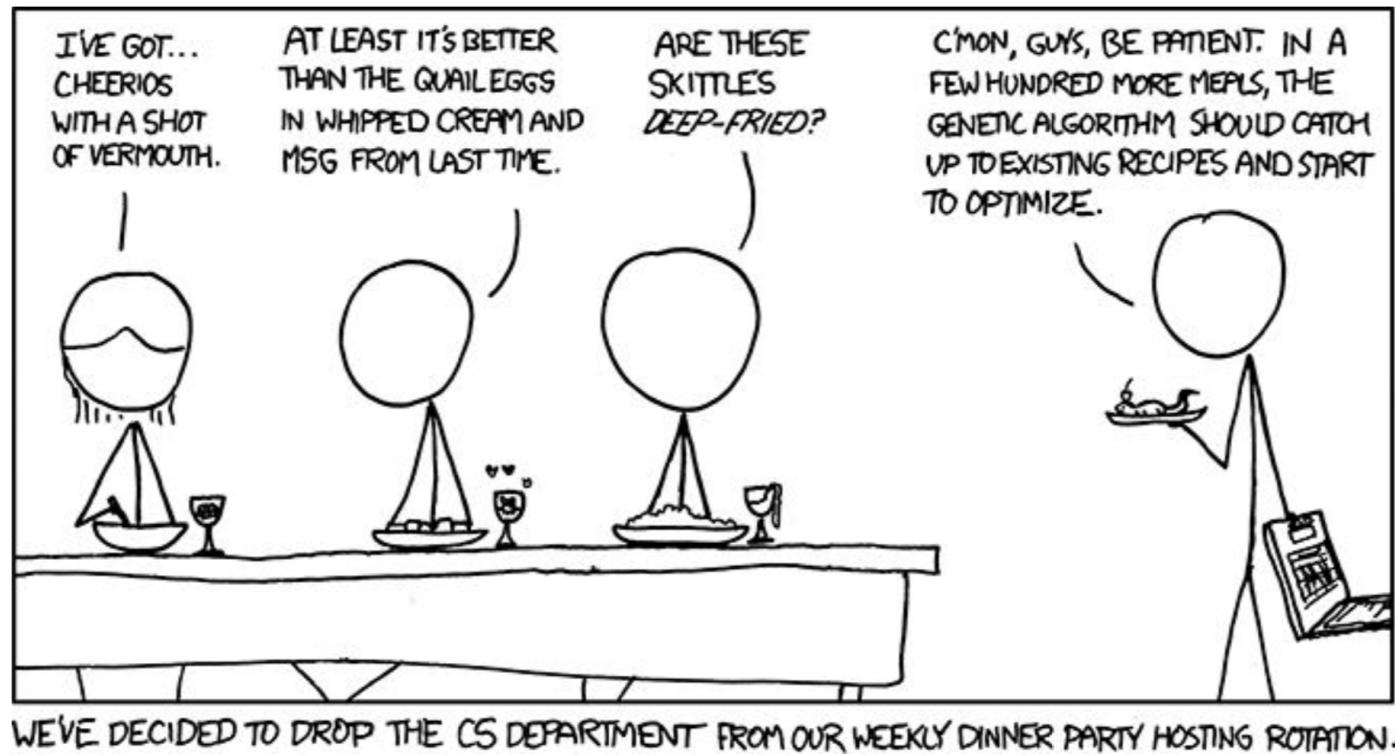


# CSCI 3202: Intro to Artificial Intelligence

## Lecture 11: Optimization, Local Search, & Genetic Algorithms

---

Rachel Cox  
Department of  
Computer Science



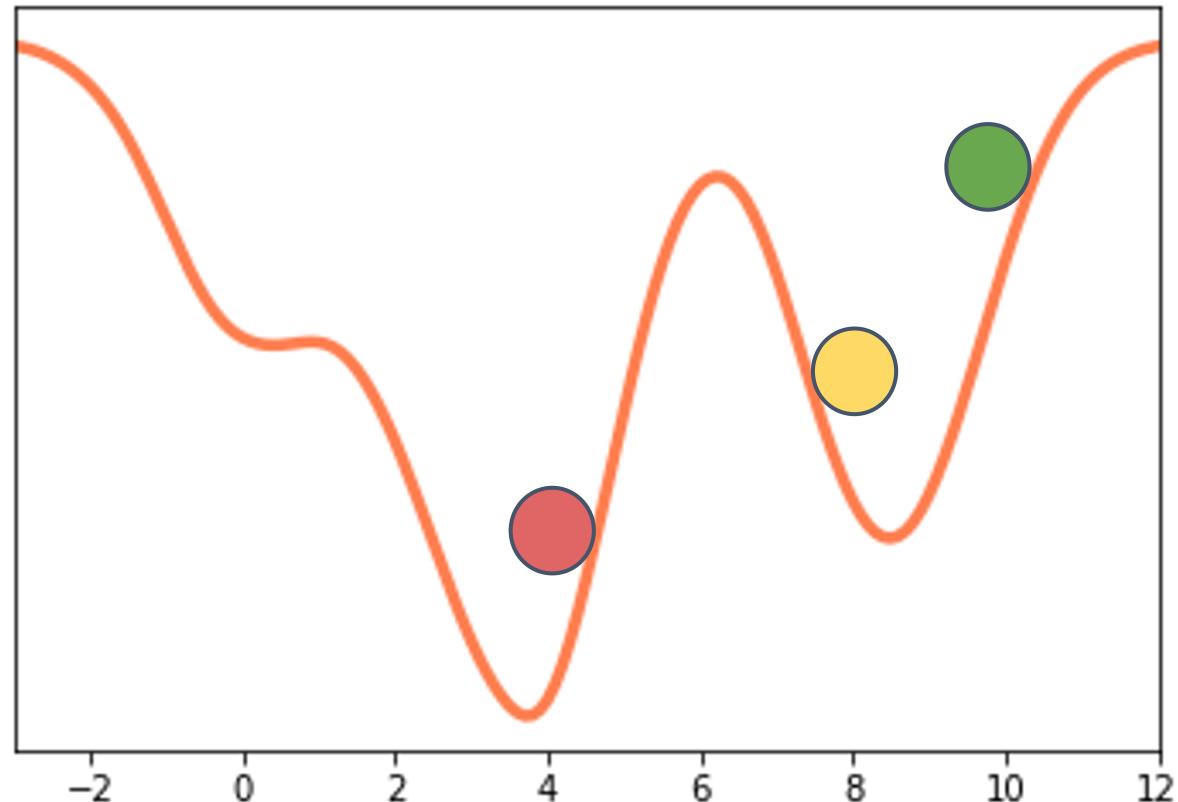
# Simulated Annealing

---

Trying to find global **minimum** (in this case)

- Propose **random** moves
- Accept with a probability that depends on:
  - 1) time
  - 2) how “good” the move is
    - Accept move without question if it’s better than our current situation
    - Accept move with some probability  $< 1$  if it’s worse

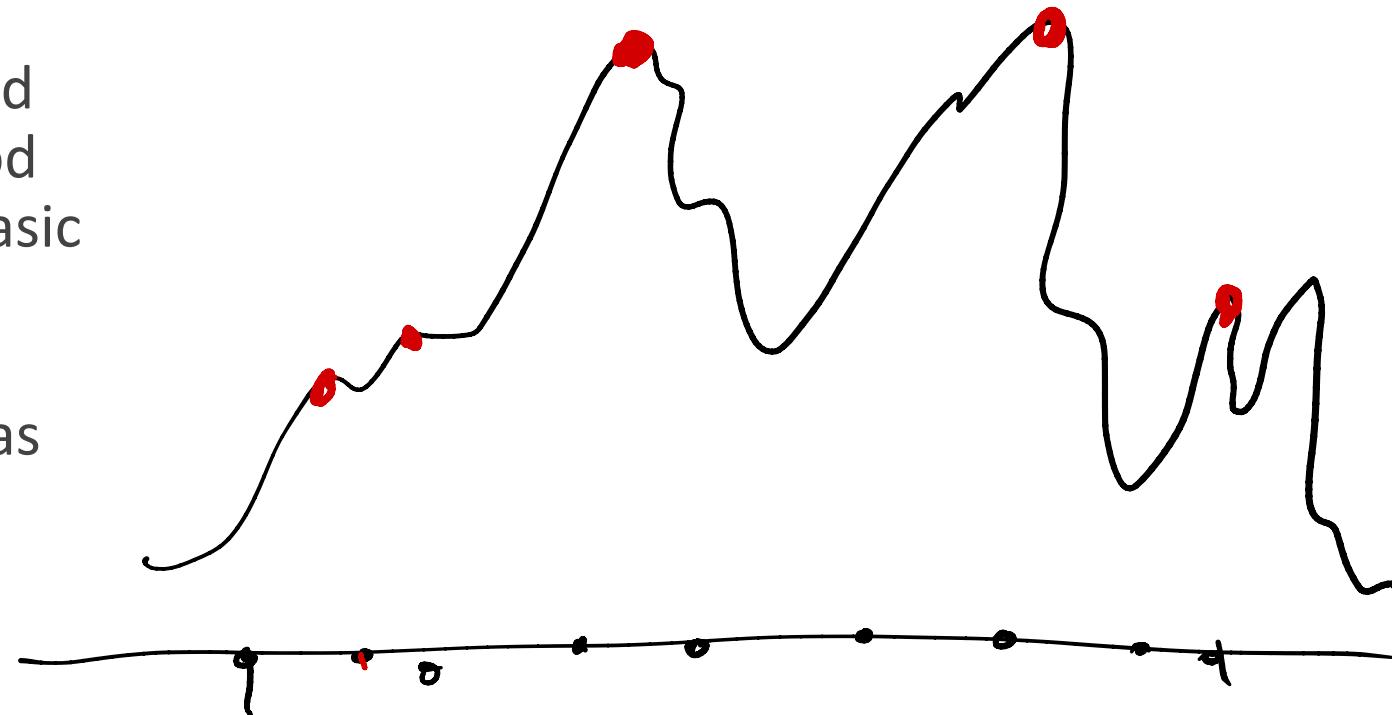
High temp - bouncing around  
Low temp - hill climbing  
Middle temp - hill climbing AND bouncing



# Random Restart Hill-Climbing

---

- Conduct a series of hill-climbing searches with a randomized starting point.
- If your random restart initial points end up all being close together, this method may not be any more effective than basic hill-climbing.
- You want to cover the space as much as you can.
- Cost is a constant multiple of hill-climbing - not much more expensive.



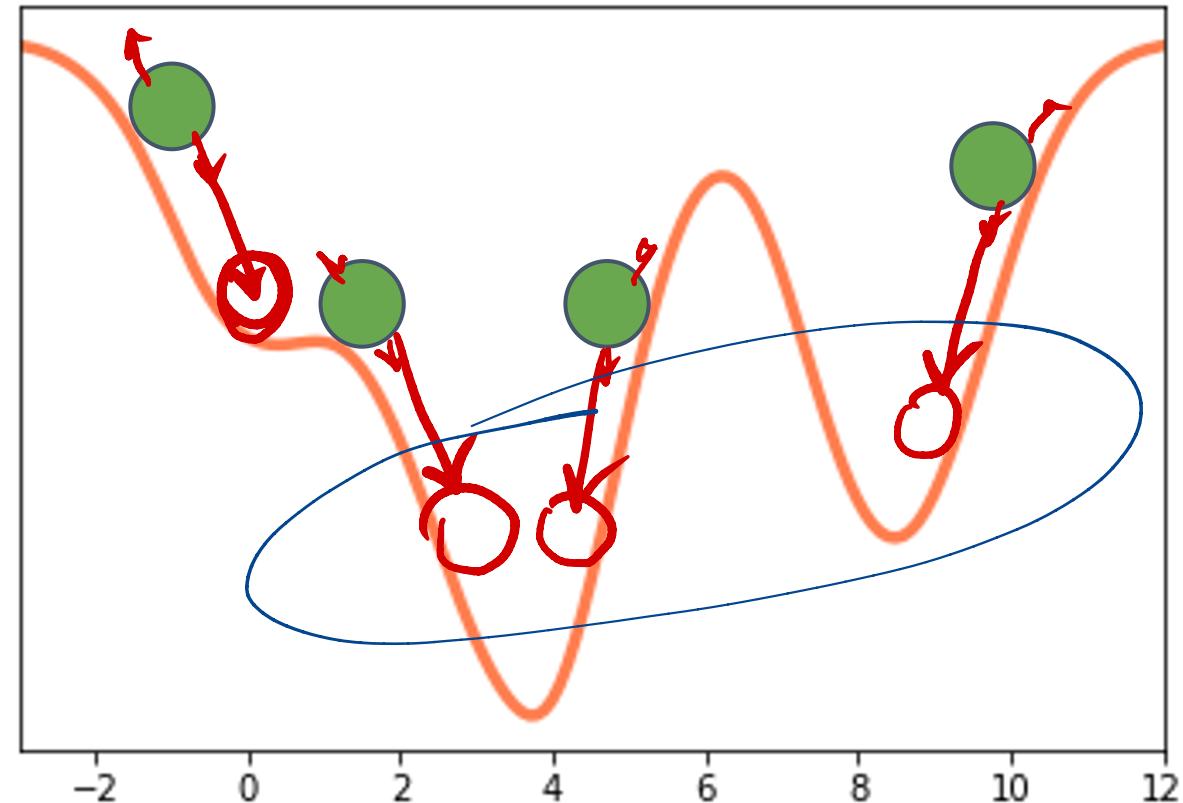
# Local Beam Search

Fire off  $k$  states to explore

Each step:

- Propose ***all*** successors to each of the  $k$  states
- Pick  $k$  of them to keep exploring in the next step
  - Can be **best  $k$** , or **random** (possibly with some performance-weighting)
- Think of these as **generations** of states...

↓  
Suppose this is  
minimization



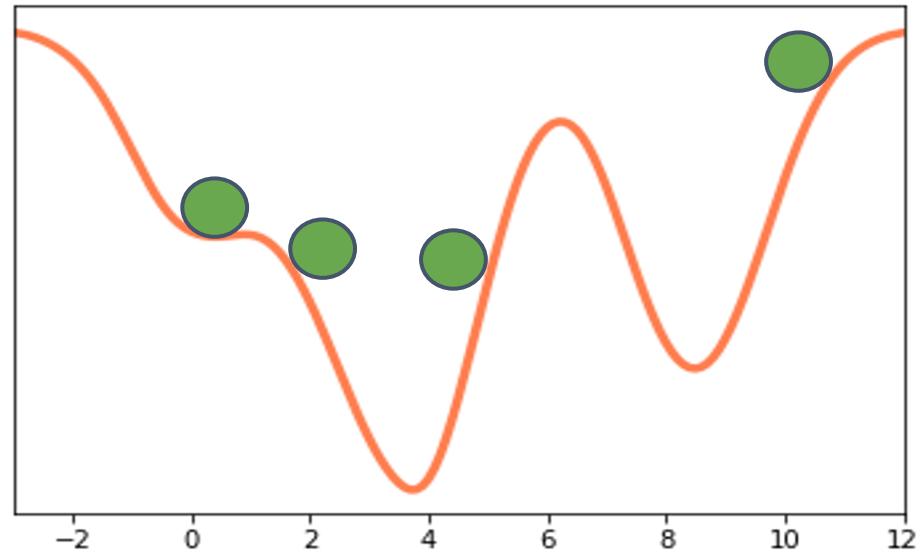
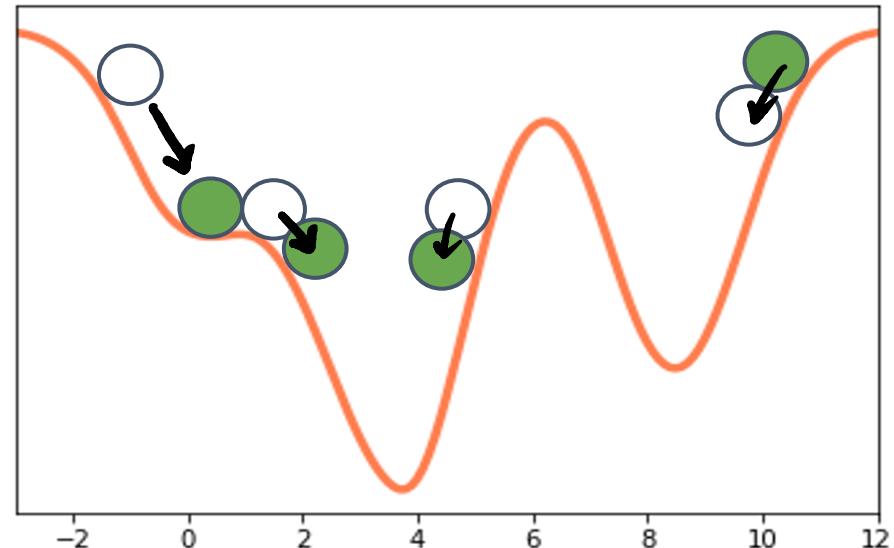
# Local Beam Search

---

Fire off  $k$  states to explore

Each step:

- Propose ***all*** successors to each of the  $k$  states
- If any of successors is goal -> finished
- Pick  $k$  of them to keep exploring in the next step
  - Can be **best  $k$** , or **random** (possibly with some performance-weighting)
- Think of these as **generations** of states...

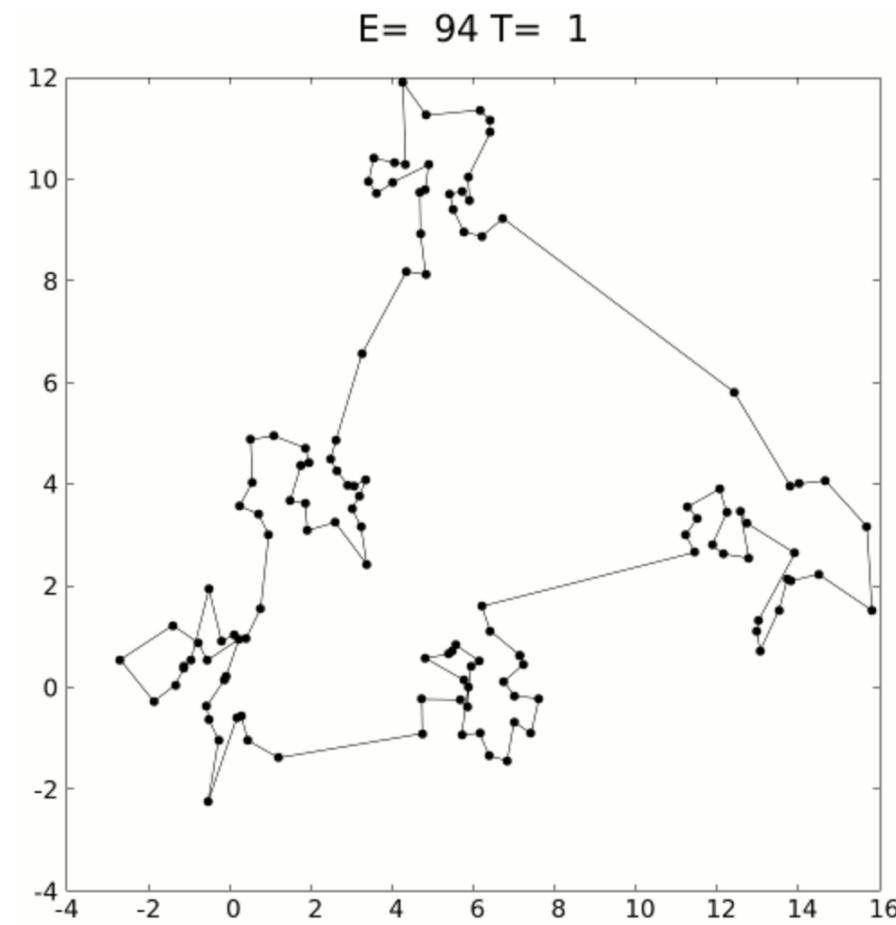
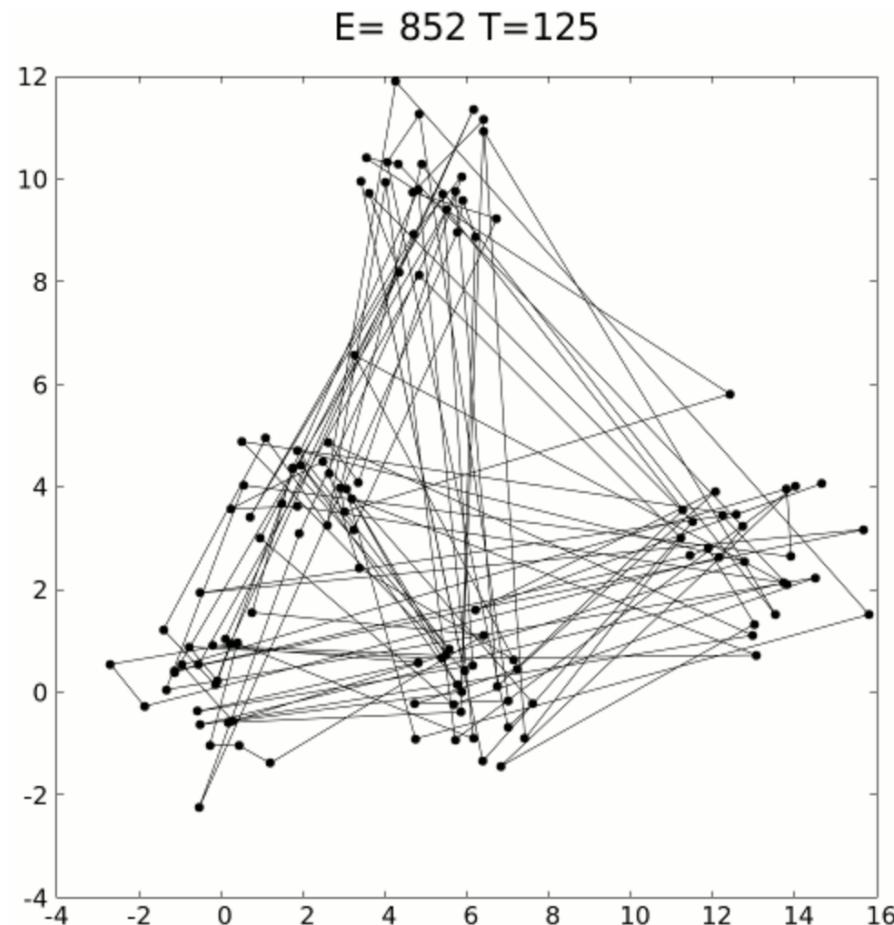


# Applications

---

Traveling Salesman Problem - Each state is a permutation of the cities visited

- Neighbors are set of permutations produced by reversing order of any two successive cities.
- Attempt to improve solution by iteratively improving its parts.



# Genetic Algorithms

---

"The fact that life evolved out of nearly nothing, some 10 billion years after the universe evolved out of literally nothing, is a fact so staggering that I would be mad to attempt words to do it justice." - Richard Dawkins

Darwinian Natural Selection: 3 key principles that must be in place for evolution to happen:

- 1) Heredity - process by which children receive the properties of their parents
- 2) Variation - Variety of traits present in population or a means with which to introduce variation
- 3) Selection - Some members have opportunity to pass on genetic info and some don't; survival of the fittest

# Genetic Algorithms

## SETUP

Step 1: **Initialize:** Create a population of N elements, each with randomly generated DNA.

## DRAW

Step 2: **Selection:** Evaluate the fitness of each element of the population and build a mating pool.

Step 3: **Reproduction:** Repeat N times:

- a. Pick two parents with probability according to relative fitness.
- b. Crossover — create a “child” by combining the DNA of these two parents.
- c. Mutation — mutate the child’s DNA based on a given probability.
- d. Add the new child to a new population.

Step 4. Replace the old population with the new population and return to Step 2.

*variation*

} "survival of  
fittest"  
aka selection

*heredity*

# Genetic Algorithms

Variant of stochastic (local) beam search in which successor states are generated by combining (*breeding*) two predecessor (*parent*) states.

Initialization:

- Begin with **population** of  $k$  randomly generated states
- Each **individual** is represented as a **string** (like DNA) from some finite alphabet

Member
28439198
87932185
12197835
90271049

population of 4  
"State"      string of 8 "genes"  
gene : digit from 0 to 9      analogous to DNA  
"g"    "t"    "a"  
       "c"

# Genetic Algorithms

$$\text{goal?} \rightarrow \frac{10}{10+15+25+6} = \frac{10}{56} \times .10$$

One generation:

- Calculate everyone's "fitness" for reproduction based on objective function values
- Yields probability of being chosen for reproduction
- Pick individuals from population for reproduction (some can be picked multiple times)
- Create new members' genes by **crossing over** parents' DNA
- Add in **mutations** with (small) independent probability

$\xrightarrow{\text{one way for reproductive prob:}}$

$$\frac{\text{fitness}_i}{\sum \text{fitness}}$$

Member	Fitness	Repr. prob.	Selection	Crossover	Mutation
28439198	10	18%	12197835	12197185	13197185
87932185	15	27%	87932185	87932835	87932035
12197835	25	45%	12197835	28437835	28437835
90271049	6	10%	28439198	12199198	92199198

*new population* 10

# Genetic Algorithms

---

```
def genetic_algorithm(problem, some number of generations):

    for some number of generations:

        # Create a new generation by creating a same-sized population
        # of children by:

            # 1. select for reproduction
            #     a) calculate each population member's fitness for reproduction
            #     b) calculate probability of each member reproducing
            #     c) select two mates from population based on reproductive probabilities

            # 2. mate the two individuals, creating a child in the new generation
            #     a) pick where the parents' "DNA" is spliced together

            # 3. child has a gene mutated with some small probability

        # Check whether any member satisfies the fitness goal
        # a) If yes, return that member and exit
        # b) If no, continue

    # If we've reached the end (# generations), return some failure warning
```

# Next Time

---

- *Games!*