Genetic Algorithms

ITP 435

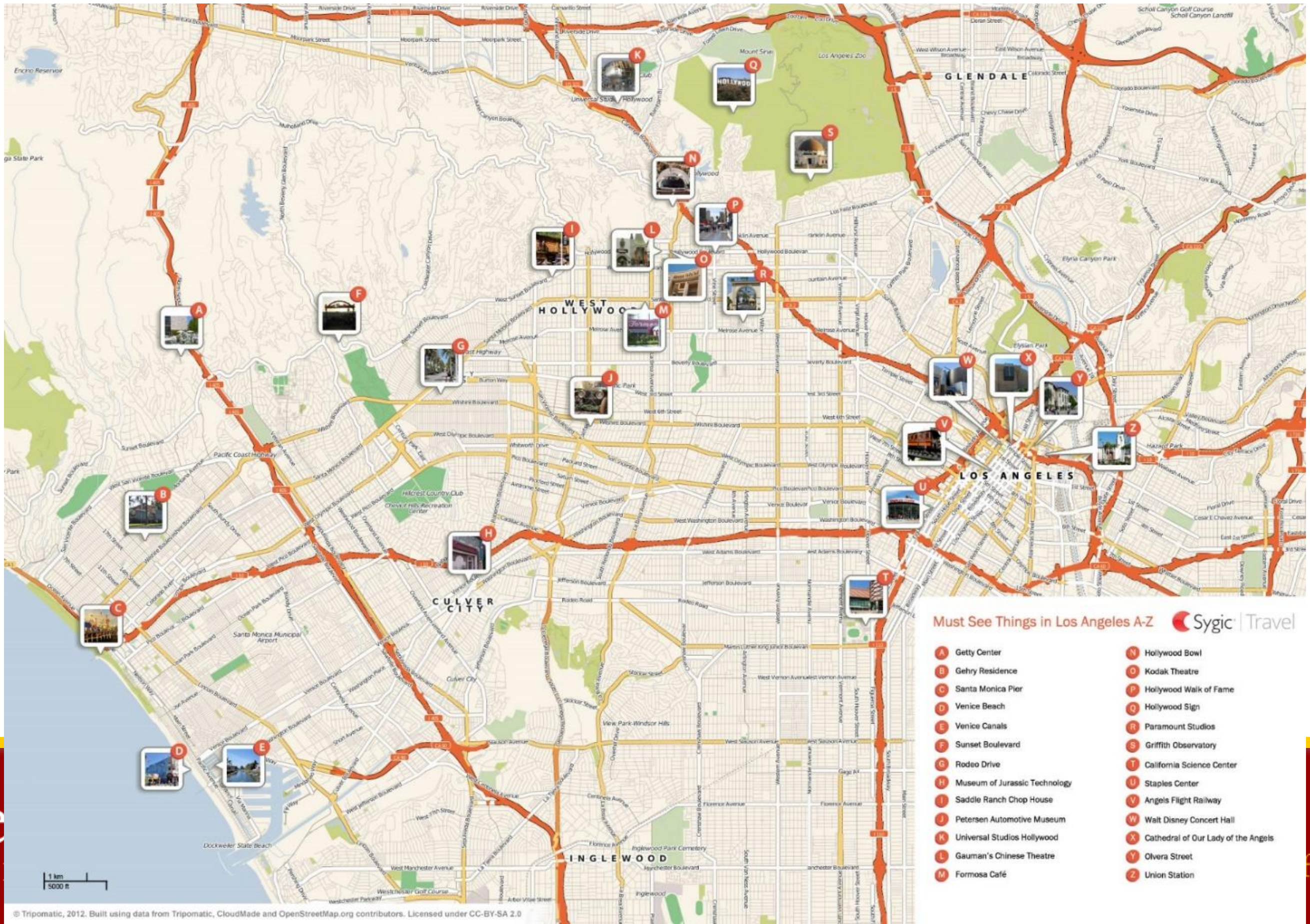
Week 3, Lecture 1

Lecturer: Sanjay Madhav

Why Genetic Algorithms?

A Problem Statement – “Travelling Salesperson”

- Given LAX + 19 different locations in Los Angeles
- Find the shortest path that starts at LAX, visits each other location exactly once, and then returns to LAX (a ***tour***)
- Assume we can fly with a helicopter (roads add complexity)


GLENDALE
WEST HOLLYWOOD
LOS ANGELES
CULVER CITY
INGLEWOOD
Must See Things in Los Angeles A-Z
Sygic Travel
A Getty Center
B Gehry Residence
C Santa Monica Pier
D Venice Beach
E Venice Canals
F Sunset Boulevard
G Rodeo Drive
H Museum of Jurassic Technology
I Saddle Ranch Chop House
J Petersen Automotive Museum
K Universal Studios Hollywood
L Gauman's Chinese Theatre
M Formosa Café
N Hollywood Bowl
O Kodak Theatre
P Hollywood Walk of Fame
Q Hollywood Sign
R Paramount Studios
S Griffith Observatory
T California Science Center
U Staples Center
V Angels Flight Railway
W Walt Disney Concert Hall
X Cathedral of Our Lady of the Angels
Y Olvera Street
Z Union Station
1 km
5000 ft
© Tripomatic, 2012. Built using data from Tripomatic, CloudMade and OpenStreetMap.org contributors. Licensed under CC-BY-SA 2.0

Travelling Trojan – Locations, Longitude/Latitude

```
LAX Airport,33.941845,-118.408635
Tommy Trojan,34.020547,-118.285397
Coliseum,34.014156,-118.287923
Chinese Theatre,34.102021,-118.340946
Whiskey a Go Go,34.090839,-118.385725
Getty Center,34.078062,-118.473892
Getty Villa,34.045868,-118.564850
Disneyland,33.812110,-117.918921
The Huntington Library,34.129178,-118.114556
Rose Bowl,34.161373,-118.167646
Griffith Observatory,34.118509,-118.300414
Hollywood Sign,34.134124,-118.321548
Magic Mountain,34.425392,-118.597230
Third Street Promenade,34.016297,-118.496838
Venice Beach,33.985857,-118.473167
Catalina Island,33.394698,-118.415119
Staples Center,34.043097,-118.267351
Dodger Stadium,34.072744,-118.240594
La Brea Tar Pits,34.063814,-118.355466
Zuma Beach,34.015489,-118.822160
```

Brute force?

- ***Idea***: Brute force and try every possible permutation
- ***Problem***: There are a lot of possible tours, 19! in this case
  - 19! is a big number
  - Suppose I can test one billion tours/second…

How big?

- If I can test one billion tours/second, and keep running nonstop…

19! /

(1,000,000,000 tours/sec) /

(60 seconds/minute) /

(60 minutes/hour) /

(24 hours/day) /

(365.25 days/year) =

How big?

- If I can test one billion tours/second, and keep running nonstop…

19! /

(1,000,000,000 tours/sec) /

(60 seconds/minute) /

(60 minutes/hour) /

(24 hours/day) /

(365.25 days/year) =

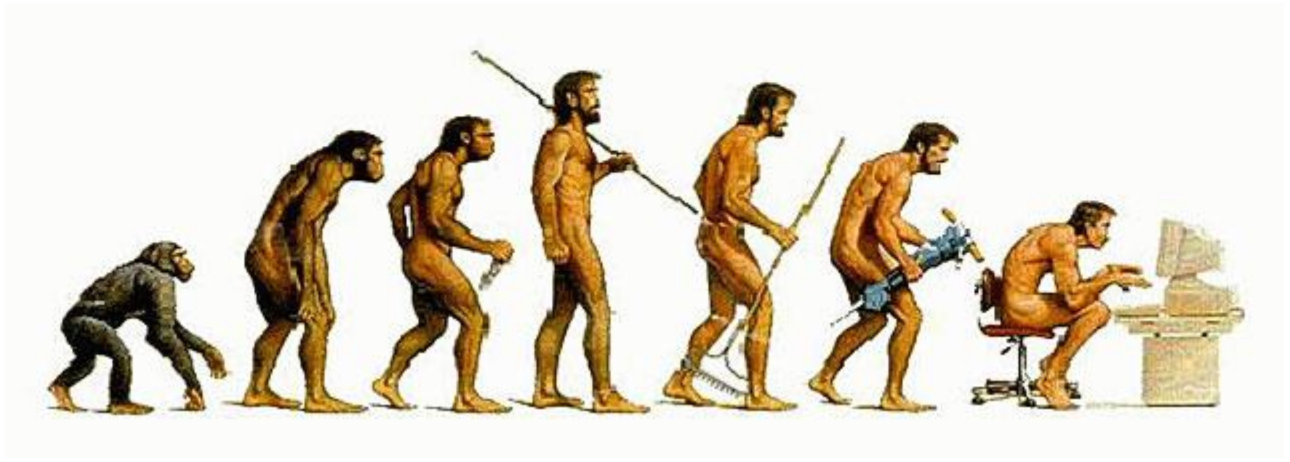
~3.85 years

- (And one billion tours/second is pretty fast!)

A more efficient algorithm?

- We don't know a more efficient algorithm that can guarantee the optimal solution
- Instead use a ***heuristic algorithm*** – try to find a solution that we think is pretty good (but can't prove how good it is)
- A ***genetic algorithm*** (GA) is one type of heuristic algorithm

Genetic Algorithms

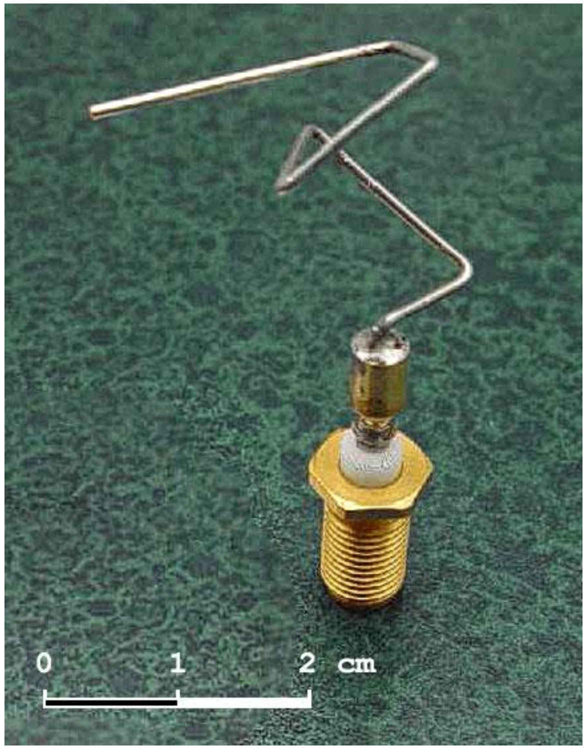


- Takes the idea of Darwinism and applies it as a heuristic algorithm

- Roughly, we start with some guesses, we try to pick the “fittest” guesses to evolve into better guesses
- After enough iterations, we stop and have a pretty good solution

Other examples – NASA designing an antenna

- NASA used a GA to design a better antenna for a spacecraft

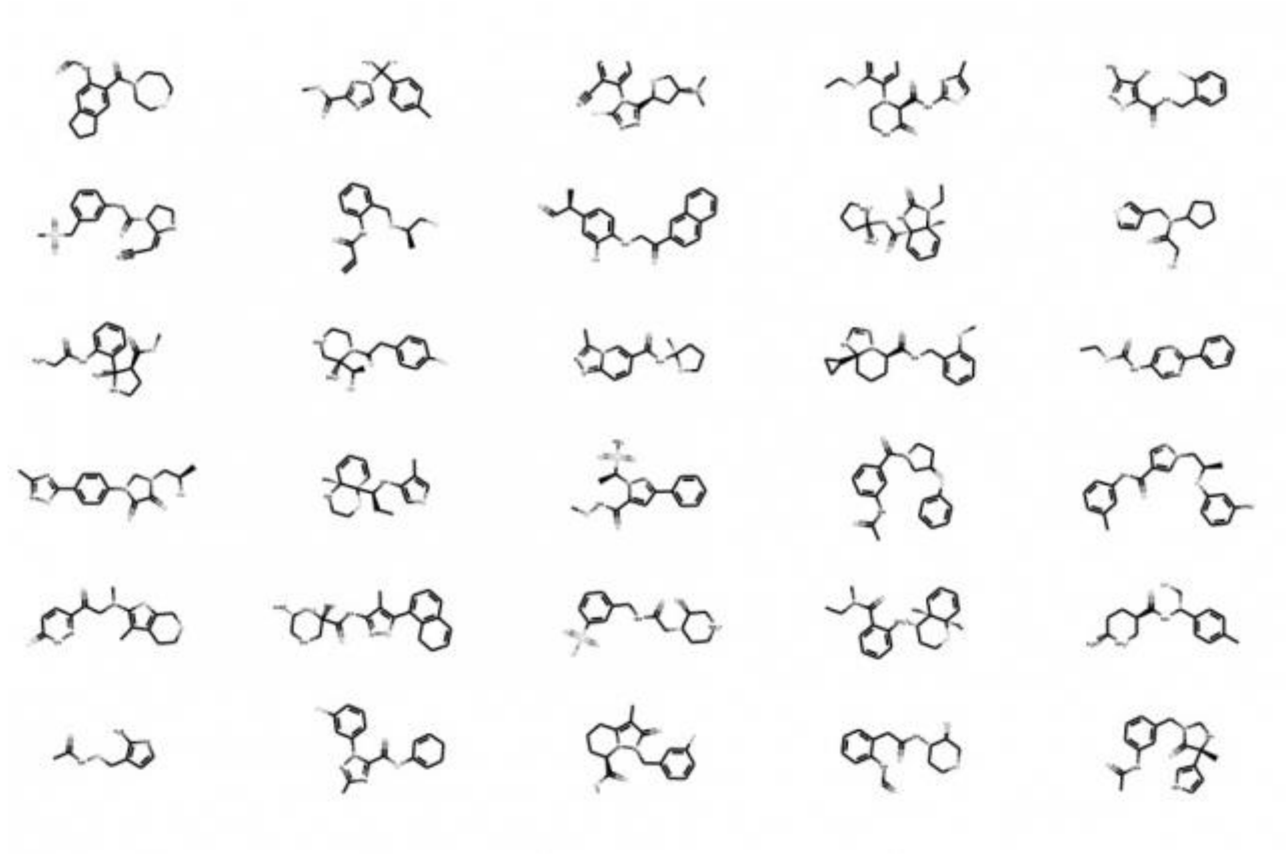


Other examples – Car aerodynamics

- Can use a GA to improve the aerodynamics to reduce drag (and increase speed)

Other examples – Drug designs

- Pharmaceutical companies increasingly use GAs and other methods to come up with new drug molecules



When are GAs a good option?

- ❑ You have a problem where you want the "best possible" solution
- ❑ There's no known efficient algorithm to find the optimal solution
- ❑ You can quantify how good or bad a solution is
- ❑ You have a way to combine parts of two (or more) solutions

What about for our case?

- ❑ You have a problem where you want the “best possible” solution
- ❑ There’s no known efficient algorithm to find the optimal solution
- ❑ You can quantify how good or bad a solution is
- ❑ You have a way to combine parts of two (or more) solutions

What about for our case?

- [x] You have a problem where you want the "best possible" solution

    We want the "best tour"

- [ ] There's no known efficient algorithm to find the optimal solution

- [ ] You can quantify how good or bad a solution is

- [ ] You have a way to combine parts of two (or more) solutions

What about for our case?

☑ You have a problem where you want the “best possible” solution

We want the “best tour”

☑ There’s no known efficient algorithm to find the optimal solution

Brute force is n!

☐ You can quantify how good or bad a solution is

☐ You have a way to combine parts of two (or more) solutions

What about for our case?

- ☑ You have a problem where you want the “best possible” solution

  We want the “best tour”

- ☑ There’s no known efficient algorithm to find the optimal solution

  Brute force is n!

- ☑ You can quantify how good or bad a solution is

  Better solutions have shorter paths

- ☐ You have a way to combine parts of two (or more) solutions

What about for our case?

☑ You have a problem where you want the "best possible" solution

We want the "best tour"

☑ There's no known efficient algorithm to find the optimal solution

Brute force is n!

☑ You can quantify how good or bad a solution is

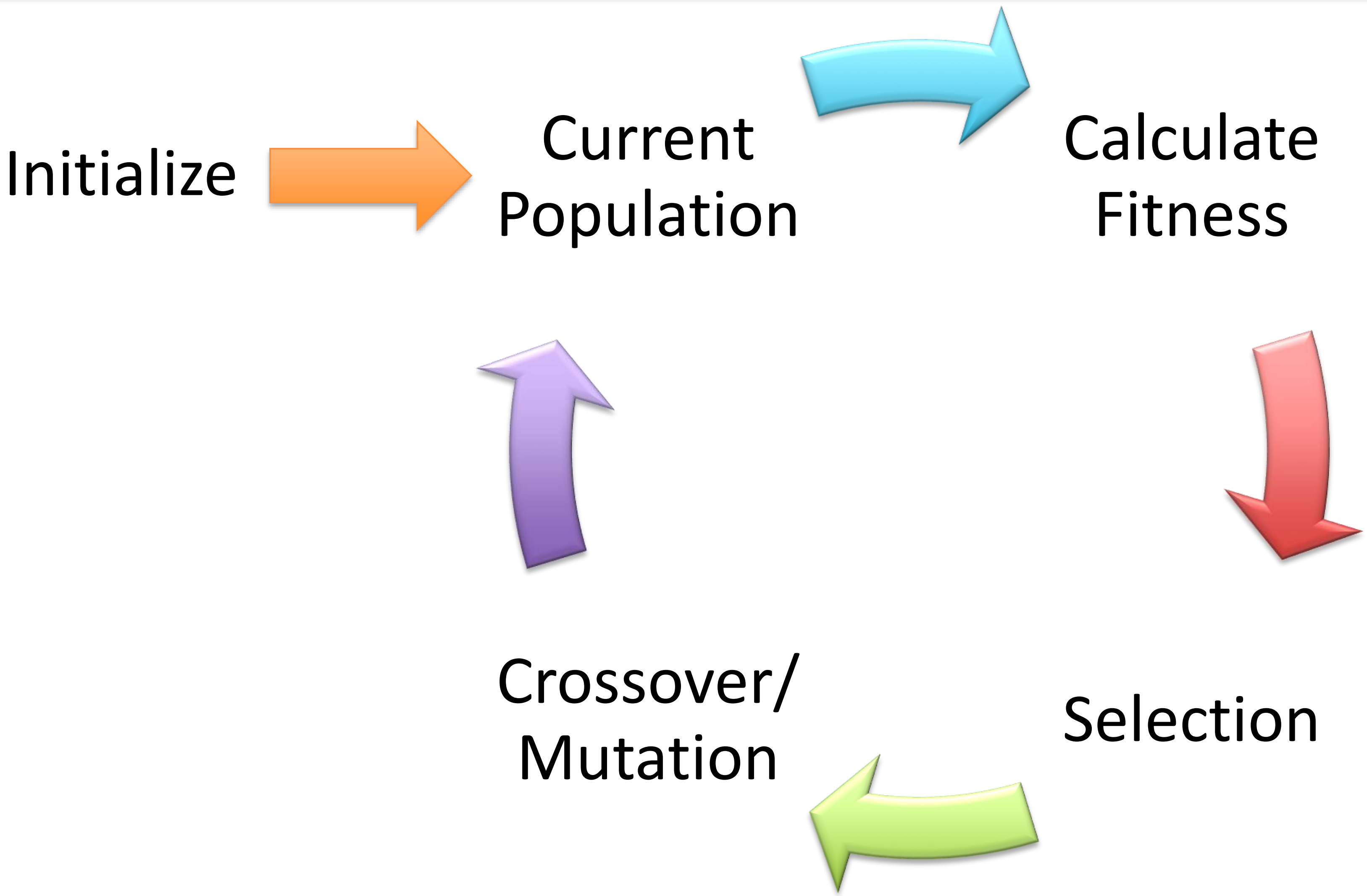
Better solutions have shorter paths

☑ You have a way to combine parts of two (or more) solutions

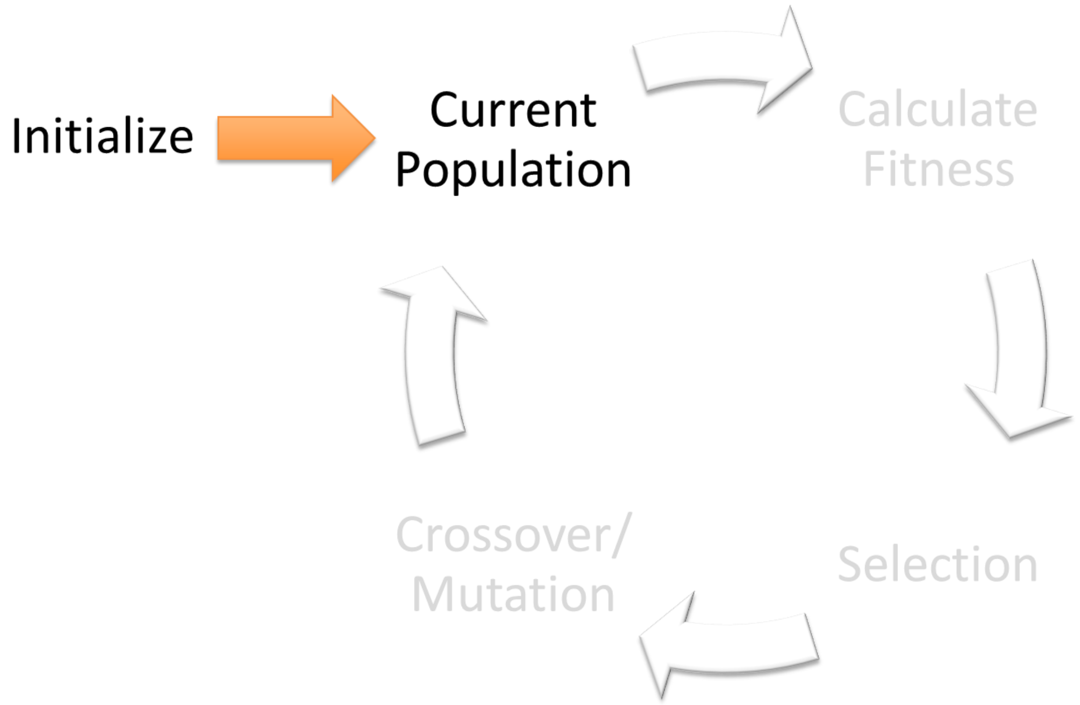
We can take portions of multiple tours (more about this soon…)

Genetic Algorithm Cycle

Overview


Initialize
Current Population
Calculate Fitness
Selection
Crossover/ Mutation

Initialize GA



- Generate a random initial population
- Ideally want a simple representation of each member (text "genomes" are popular, but not the only way)
- Size of population varies (for us, a command line argument)

Our Population “Genome” Representation

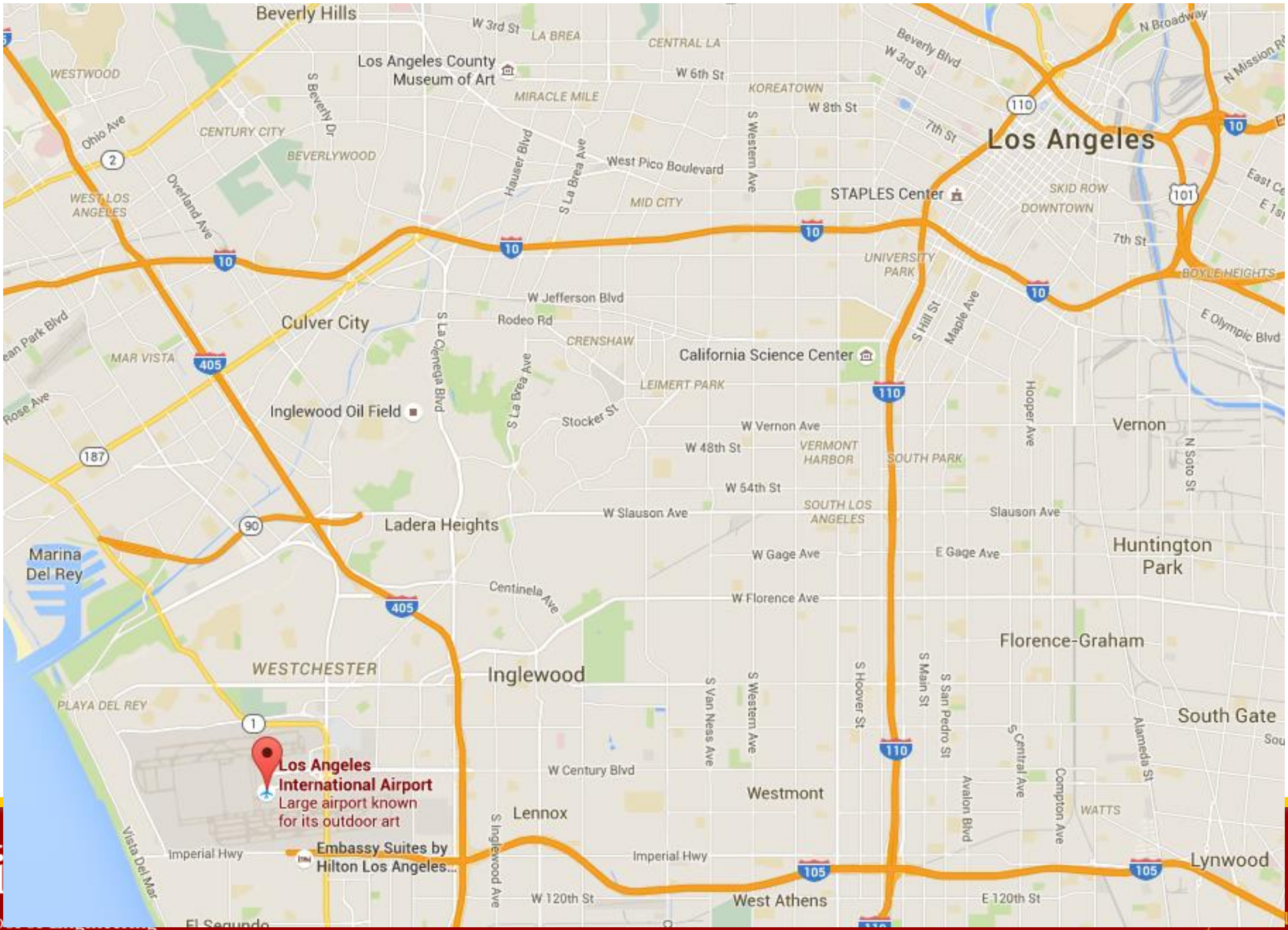
- A vector of numbers corresponding to the index of the location (the first location in the file is 0)
- So this…

0	5	3	2	1	4

- Means first start at location 0, then go to 5, then go to 3, then go to 2, then go to 1, then go to 4, then return to 0 (implied)

Location 0 is special in our case

- We always start/end at location 0 (LAX in this example), so every valid path will have index 0 as the first location

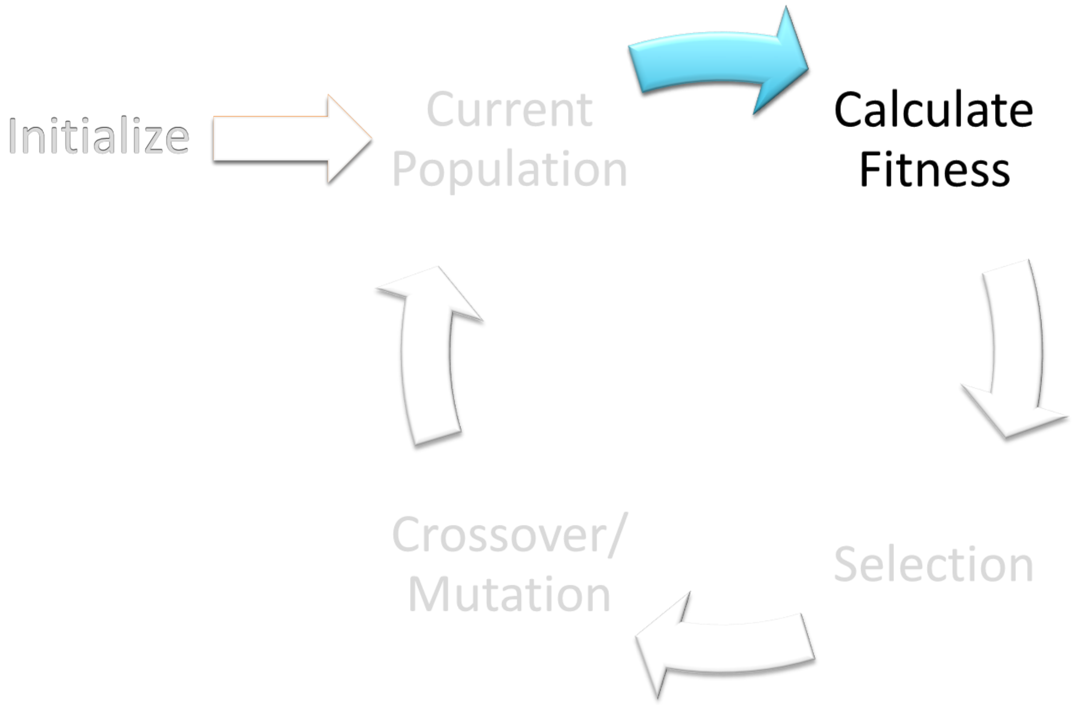

Los Angeles International Airport
Large airport known for its outdoor art
Los Angeles
Inglewood
Culver City
Beverly Hills

Sample Initial Population (Size = 8)

#	Population					
0	0	5	3	2	1	4
1	0	2	4	5	3	1
2	3	5	2	4	0	1
3	1	3	2	4	5	0
4	5	4	0	2	1	3
5	5	2	0	1	4	3
6	5	0	2	4	3	1
7	0	1	3	2	4	5

Note: In this case, we assume the tour can start at any location. In our "travelling Trojan" problem we will always start at location 0

Calculate Fitness



- Calculate the fitness function for each member of the population
- For us, lower fitness = shorter distance = better

Fitness Function

- For fitness, we use the ***Haversine distance*** formula to compute the distance of each segment on the path:

```
dlon = lon2 - lon1
dlat = lat2 - lat1
a = (sin(dlat/2))² + cos(lat1) * cos(lat2) * (sin(dlon/2))²
c = 2 * atan2( sqrt(a), sqrt(1-a) )
distance = 3961 * c
```

- Once we have the segment distances, sum them together for the total tour distance

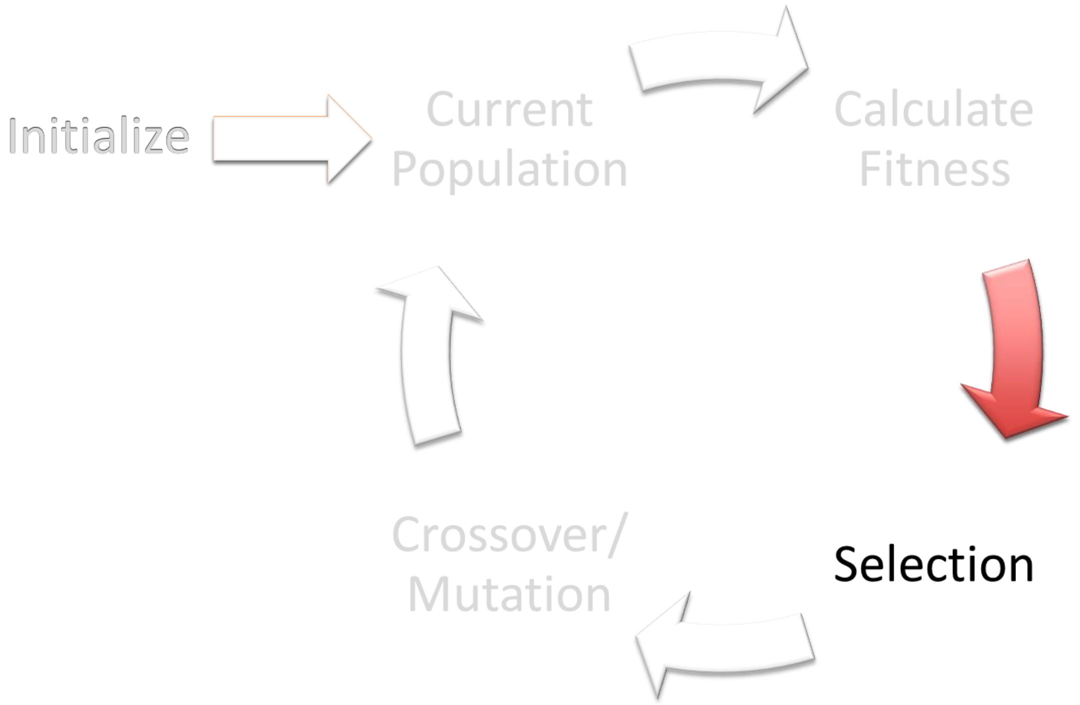
- ***PA Note:*** You have to convert the numbers in the input file from degrees to radians. To convert, multiply by 0.0174533

Sample Calculate Fitness

#	Population						Fitness
0	0	5	3	2	1	4	75 mi
1	0	2	4	5	3	1	85 mi
2	3	5	2	4	0	1	80 mi
3	1	3	2	4	5	0	70 mi
4	5	4	0	2	1	3	90 mi
5	5	2	0	1	4	3	87 mi
6	5	0	2	4	3	1	91 mi
7	0	1	3	2	4	5	92 mi

In this example, population member #3 is fittest (shortest tour)

Selection



- Based on fitness rankings, select pairs of individuals to reproduce
- Should give some (but not all) preference to fitter individuals
- Select number of pairs = population size for pairwise reproduction
- Many ways to do this, slides describe what we use in the PA

Survival of the Fittest?

Sample Selection Probabilities

#	Population						Fitness
0	0	5	3	2	1	4	75 mi
1	0	2	4	5	3	1	85 mi
2	3	5	2	4	0	1	80 mi
3	1	3	2	4	5	0	70 mi
4	5	4	0	2	1	3	90 mi
5	5	2	0	1	4	3	87 mi
6	5	0	2	4	3	1	91 mi
7	0	1	3	2	4	5	92 mi

Selection Chance

3	0	2	1	5	4	6	7

Step: 1. Sort population member #s by fitness (smallest to largest)

Sample Selection Probabilities

#	Population						Fitness
0	0	5	3	2	1	4	75 mi
1	0	2	4	5	3	1	85 mi
2	3	5	2	4	0	1	80 mi
3	1	3	2	4	5	0	70 mi
4	5	4	0	2	1	3	90 mi
5	5	2	0	1	4	3	87 mi
6	5	0	2	4	3	1	91 mi
7	0	1	3	2	4	5	92 mi

Selection Chance

3	0	2	1	5	4	6	7
1/8	1/8	1/8	1/8	1/8	1/8	1/8	1/8

Step: 2.a. Give each member the same probability (1/popSize)

Sample Selection Probabilities

#	Population						Fitness
0	0	5	3	2	1	4	75 mi
1	0	2	4	5	3	1	85 mi
2	3	5	2	4	0	1	80 mi
3	1	3	2	4	5	0	70 mi
4	5	4	0	2	1	3	90 mi
5	5	2	0	1	4	3	87 mi
6	5	0	2	4	3	1	91 mi
7	0	1	3	2	4	5	92 mi

Selection Chance

3	0	2	1	5	4	6	7
6/8	6/8	1/8	1/8	1/8	1/8	1/8	1/8

Step: 2.b. Multiply the two fittest by 6 (the 6 is an arbitrary magic number)

Sample Selection Probabilities

#	Population						Fitness
0	0	5	3	2	1	4	75 mi
1	0	2	4	5	3	1	85 mi
2	3	5	2	4	0	1	80 mi
3	1	3	2	4	5	0	70 mi
4	5	4	0	2	1	3	90 mi
5	5	2	0	1	4	3	87 mi
6	5	0	2	4	3	1	91 mi
7	0	1	3	2	4	5	92 mi

Selection Chance

3	0	2	1	5	4	6	7
6/8	6/8	3/8	3/8	1/8	1/8	1/8	1/8

Step: 2.c. Multiply the remainder of the top half, not including the top two, by 3 (3 is a magic number)

Sample Selection Probabilities

#	Population						Fitness
0	0	5	3	2	1	4	75 mi
1	0	2	4	5	3	1	85 mi
2	3	5	2	4	0	1	80 mi
3	1	3	2	4	5	0	70 mi
4	5	4	0	2	1	3	90 mi
5	5	2	0	1	4	3	87 mi
6	5	0	2	4	3	1	91 mi
7	0	1	3	2	4	5	92 mi

Selection Chance

3	0	2	1	5	4	6	7
6/8	6/8	3/8	3/8	1/8	1/8	1/8	1/8

sum = 22/8

Step: 2.d. Renormalize the probabilities (sum them and divide by sum)

Sample Selection Probabilities

#	Population						Fitness
0	0	5	3	2	1	4	75 mi
1	0	2	4	5	3	1	85 mi
2	3	5	2	4	0	1	80 mi
3	1	3	2	4	5	0	70 mi
4	5	4	0	2	1	3	90 mi
5	5	2	0	1	4	3	87 mi
6	5	0	2	4	3	1	91 mi
7	0	1	3	2	4	5	92 mi

Selection Chance

3	0	2	1	5	4	6	7
6/22	6/22	3/22	3/22	1/22	1/22	1/22	1/22

sum = 22/8

Step: 2.d. Renormalize the probabilities (sum them and divide by sum)

Sample Selection Probabilities

#	Population						Fitness
0	0	5	3	2	1	4	75 mi
1	0	2	4	5	3	1	85 mi
2	3	5	2	4	0	1	80 mi
3	1	3	2	4	5	0	70 mi
4	5	4	0	2	1	3	90 mi
5	5	2	0	1	4	3	87 mi
6	5	0	2	4	3	1	91 mi
7	0	1	3	2	4	5	92 mi

Selection Chance

3	0	2	1	5	4	6	7
6/22	6/22	3/22	3/22	1/22	1/22	1/22	1/22

sum = 22/8

Step: 2.d. Renormalize the probabilities (sum them and divide by sum)

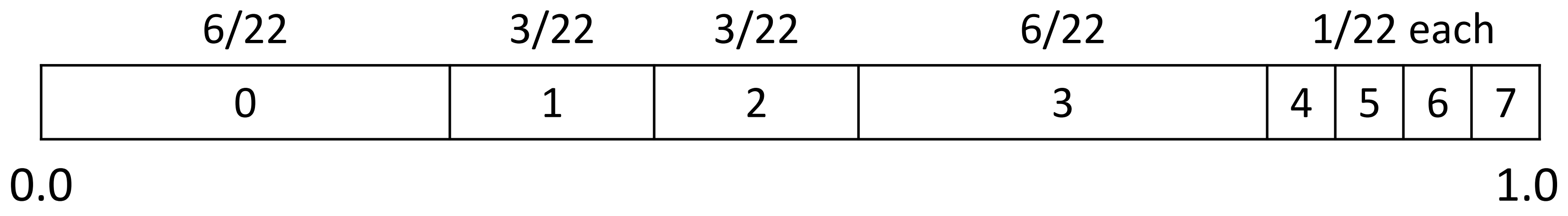
Sample Selection Probabilities

#	Population						Fitness	Selection Pr
0	0	5	3	2	1	4	75 mi	6/22
1	0	2	4	5	3	1	85 mi	3/22
2	3	5	2	4	0	1	80 mi	3/22
3	1	3	2	4	5	0	70 mi	6/22
4	5	4	0	2	1	3	90 mi	1/22
5	5	2	0	1	4	3	87 mi	1/22
6	5	0	2	4	3	1	91 mi	1/22
7	0	1	3	2	4	5	92 mi	1/22

Step: Now we have selection probabilities

Sample Selecting Pairs

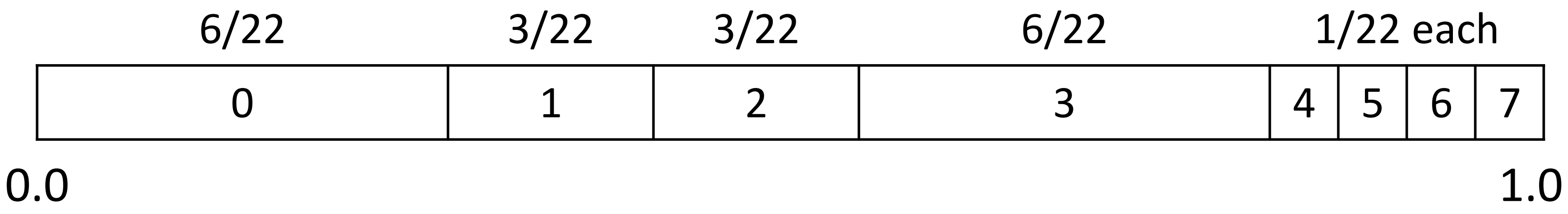
#	Pr
0	6/22
1	3/22
2	3/22
3	6/22
4	1/22
5	1/22
6	1/22
7	1/22



Step: 3. Assign the range of [0,1] to the different population members, based on probabilities

Sample Selecting Pairs

#	Pr
0	6/22
1	3/22
2	3/22
3	6/22
4	1/22
5	1/22
6	1/22
7	1/22



Random Doubles = (0.40, 0.68)

Parent Pair = (1, 3)

Step: 4. Generate two random doubles in the range [0, 1] to figure out two parents

Sample Selecting Pairs

Pairs

(1, 3)

(2, 1)

(0, 3)

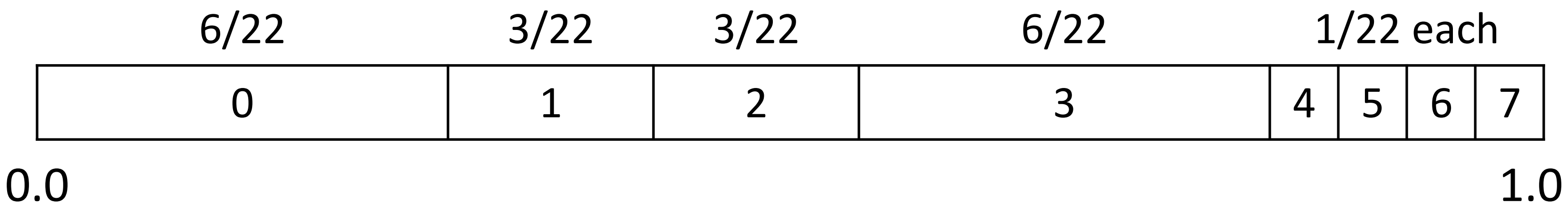
(4, 1)

(3, 5)

(3, 7)

(3, 3)

(2, 3)



Step: 5. Repeat step 4 for however many pairs you want (in our case, popSize)

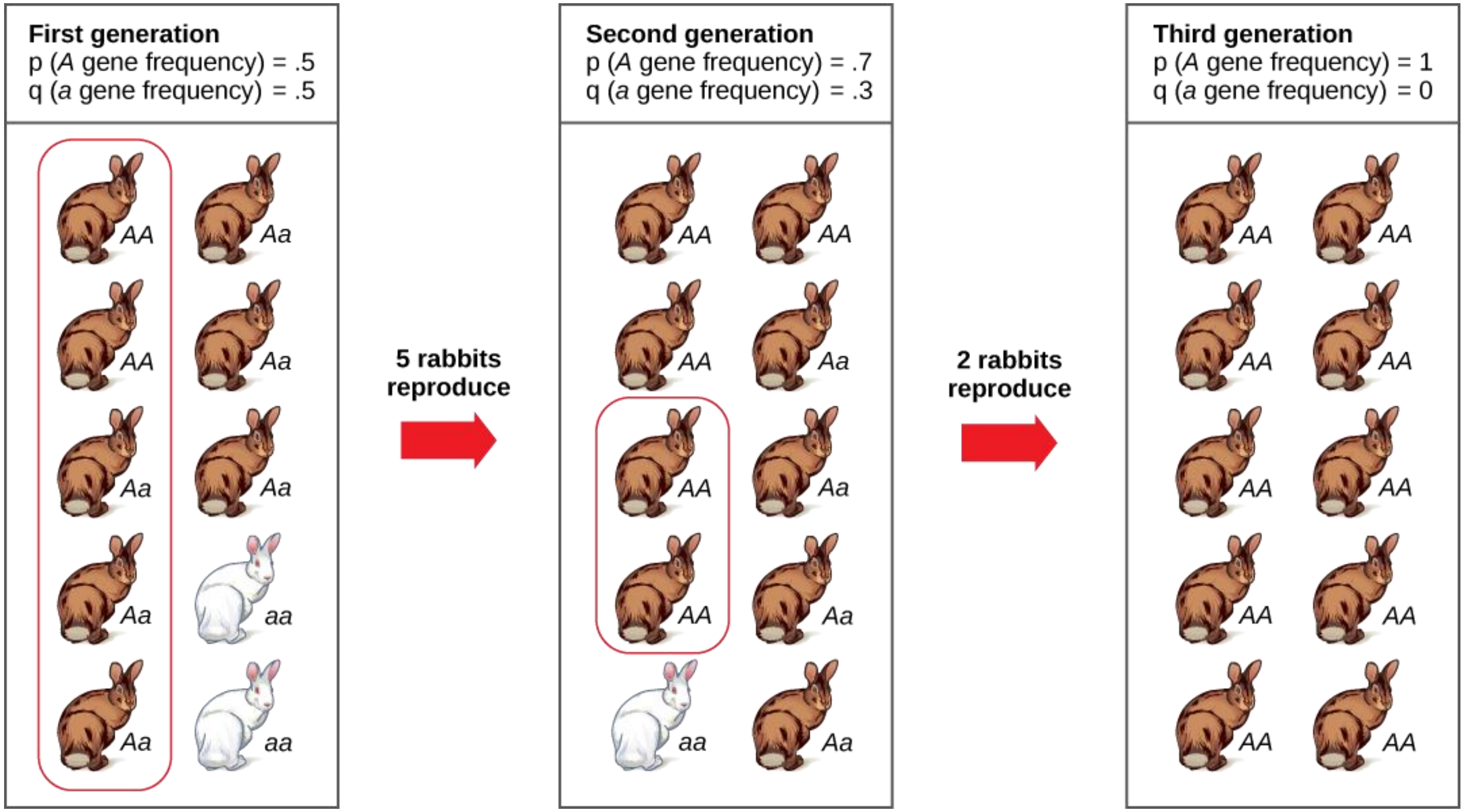
Our Selection Process

Generate the pairs as follows:

1. Sort the population by fitness
2. Distribute probability as follows:
   a. Give each member the same probability (1/popSize)
   b. Multiply the two fittest by 6
   c. Multiply the remainder of the top half, not including the top two, by 3
   d. Renormalize the probabilities
3. Assign the range of [0,1] to the different population members, based on probabilities
4. Generate two random doubles in the range [0, 1] to figure out two parents
5. Repeat step 4 for however many pairs you want (in our case, popSize)

Why not only select fittest?

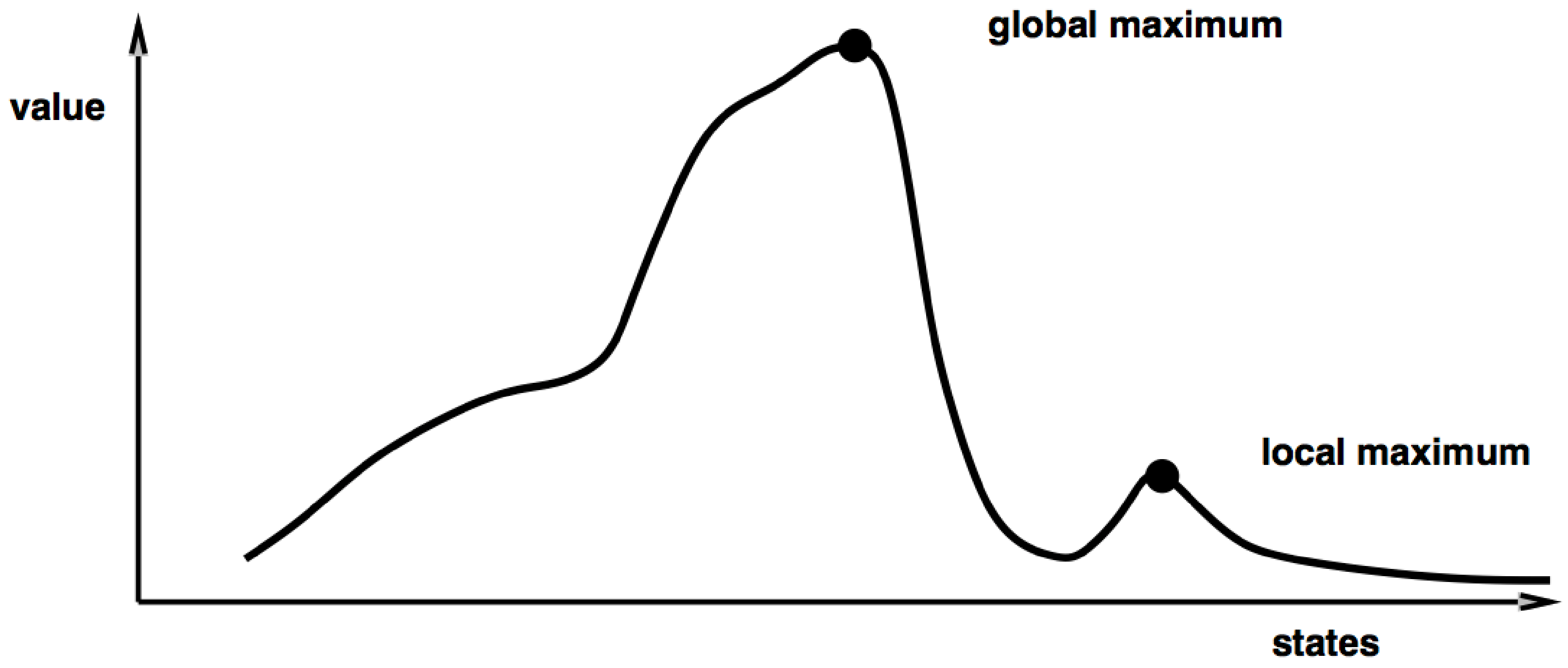
- We want to avoid rapid genetic drift



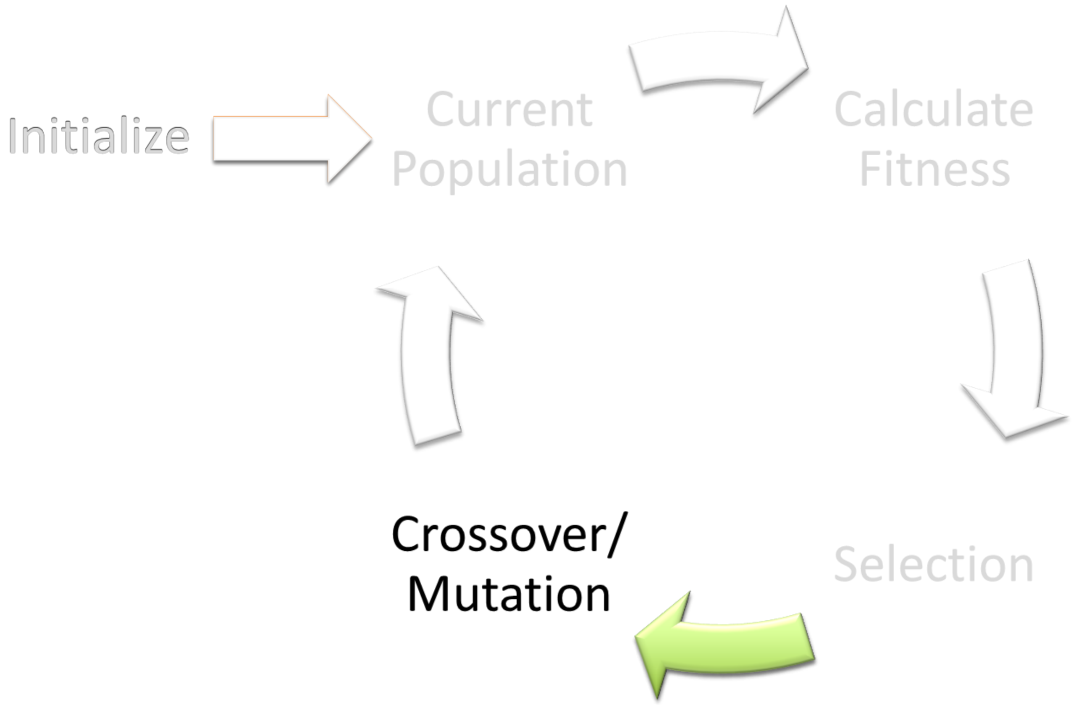
First generation
p (*A* gene frequency) = .5
q (*a* gene frequency) = .5
AA
Aa
AA
Aa
Aa
Aa
Aa
aa
Aa
aa
5 rabbits reproduce
Second generation
p (*A* gene frequency) = .7
q (*a* gene frequency) = .3
AA
AA
AA
Aa
AA
Aa
AA
Aa
aa
Aa
2 rabbits reproduce
Third generation
p (*A* gene frequency) = 1
q (*a* gene frequency) = 0
AA

Genetic Drift = Bad

- Genetic drift means we quickly converge on a solution
- However, there's no guarantee the solution is globally optimal – it could be locally optimal (classic "hill climbing" problem)



Crossover/Mutation



- Reproduce selected pairs by “crossing over” the genomes
- Might also introduce random mutations (as in real genetics)

Crossover Example

- Select whether parent A or B goes first
  - For this example, A
- Select a crossover index from 1 to size – 2
  - For this example 2

A	0	5	3	2	1	4
B	0	4	3	5	1	2
Child						

Crossover Example

- Since A goes first, and 2 is the crossover index…
- Copy from A [0…2]

A	0	5	3	2	1	4
B	0	4	3	5	1	2
Child	0	5	3			

Crossover Example

- Since B goes second
- From B, copy any locations that aren't already in the child (in order)

A	0	5	3	2	1	4
B	0	4	3	5	1	2
Child	0	5	3	4	1	2

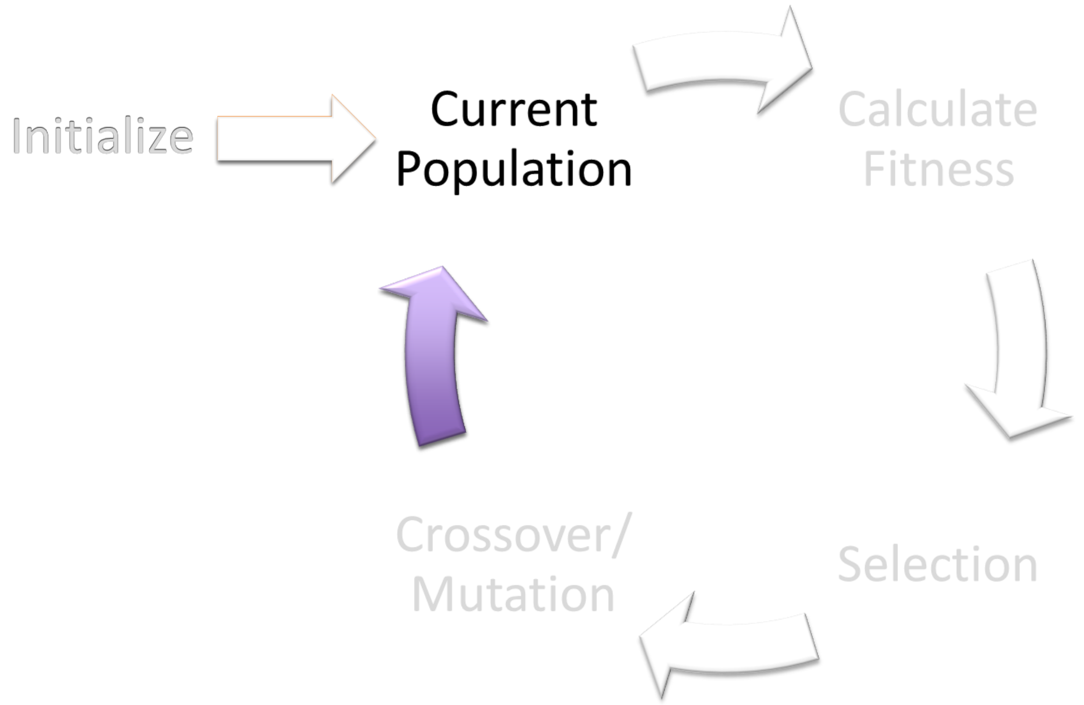
Mutations

- We'll just use a simple mutation implementation:
- There's a chance a mutation occurs when creating a child
- If we mutate, pick two random indices (not including index 0), and swap the values

Crossover Summary

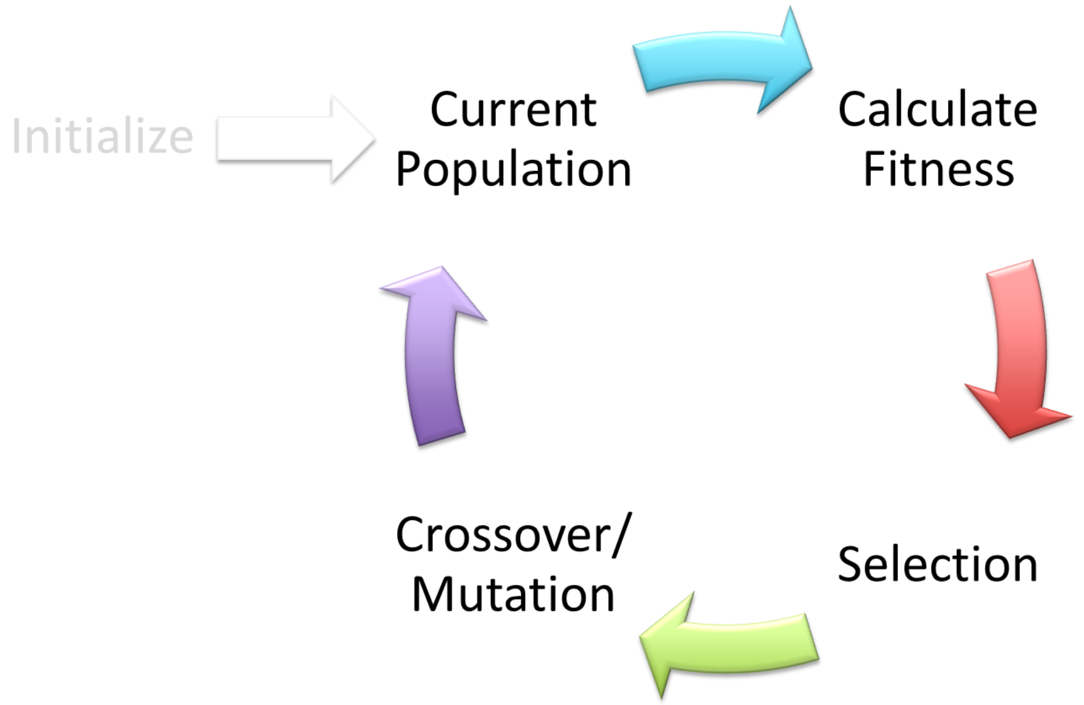
1. Generate a random crossover index value from [1, size – 2]
2. “Flip a coin” to decide which parent should go first
3. Selected first parent will copy all elements from beginning up to and including crossover index into child
4. Second parent will start at the beginning, and copy over all elements that don’t already appear in the child
5. Mutate, based on probability

Next Generation



- After calculating crossover/mutation for all pairs, we'll have a new generation for the "current population"

Repeat



- Keep repeating until reaching a condition of termination
- In our case, number of generations is specified as a command line argument

Other Ways to Shake Up the Population?

- Other approaches include adding random probabilities of cataclysmic events that wipe out a percentage of the population

Convergence in our case

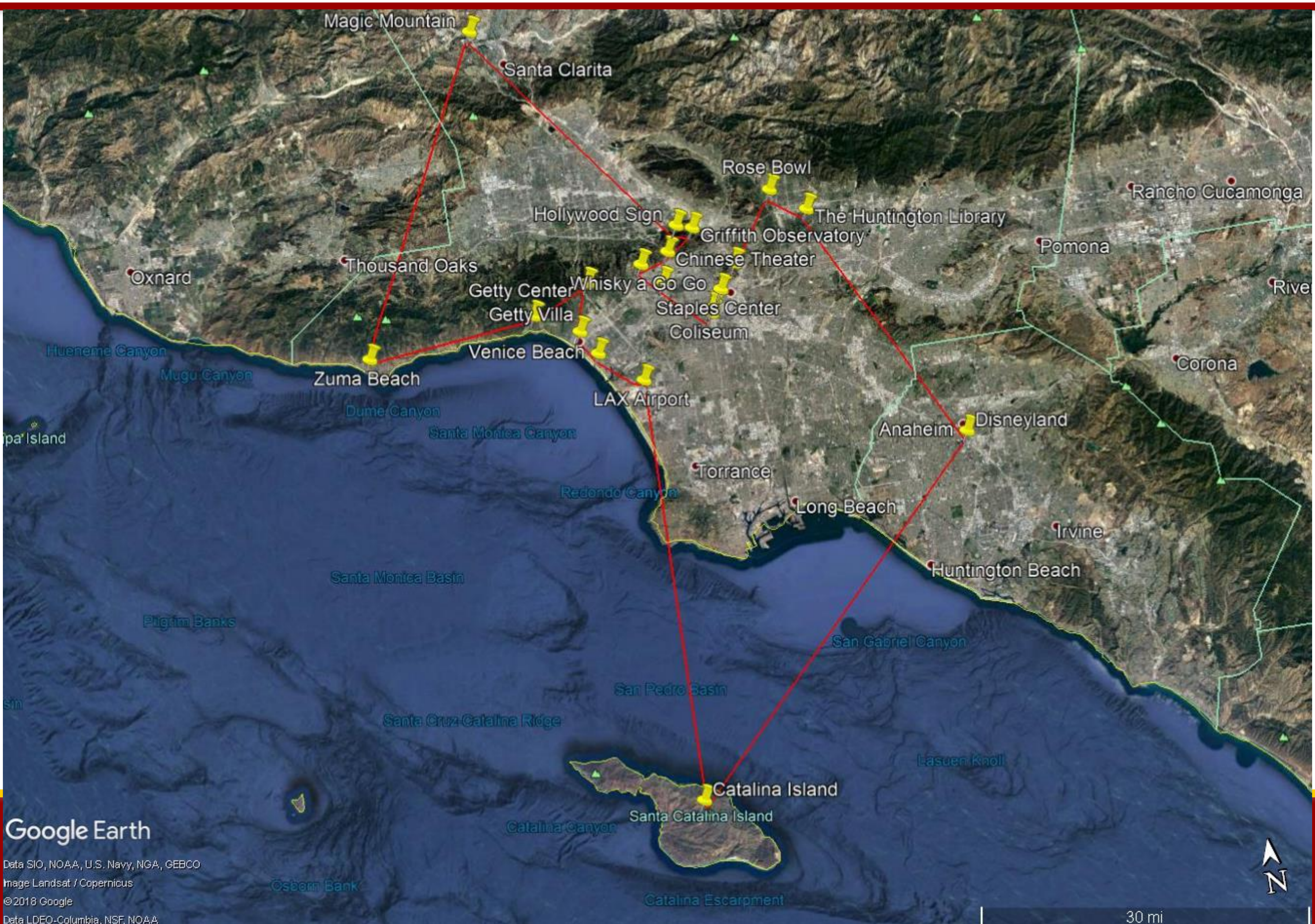
Eventually, we end up with a population with very similar members:

```
GENERATION: 200
0,3,13,5,6,19,12,11,10,14,4,18,2,1,15,16,17,9,8,7
0,14,13,5,6,19,12,11,10,3,4,18,2,1,16,17,9,8,7,15
0,14,13,5,6,19,12,11,10,3,4,18,2,1,16,17,9,8,7,15
0,6,13,5,14,19,10,12,11,3,4,18,2,1,16,17,9,8,7,15
0,14,13,5,6,19,12,11,10,3,4,18,2,1,16,17,9,8,7,15
0,7,13,5,6,19,12,11,10,3,4,18,2,1,16,17,9,8,14,15
0,14,13,5,6,19,12,11,10,3,4,18,2,1,16,17,9,8,7,15
0,14,13,5,6,12,19,11,10,3,4,18,2,1,16,17,9,8,7,15
0,14,13,5,6,19,12,11,10,3,4,18,2,1,16,17,9,8,7,15
0,4,13,5,6,19,12,11,10,3,14,18,2,1,16,17,9,8,7,15
0,14,13,5,6,19,12,11,10,3,4,18,2,1,16,17,9,8,7,15
0,14,13,5,6,19,12,11,10,3,4,18,2,1,16,17,9,8,7,15
```

Best Route?

```
LAX Airport
Venice Beach
Third Street Promenade
Getty Center
Getty Villa
Zuma Beach
Magic Mountain
Hollywood Sign
Griffith Observatory
Chinese Theatre
Whiskey a Go Go
La Brea Tar Pits
Coliseum
Tommy Trojan
Staples Center
Dodger Stadium
Rose Bowl
The Huntington Library
Disneyland
Catalina Island
LAX Airport
DISTANCE: 222.736 miles
```

Our Tour Looks Pretty Good



For a real case…

- Try different:
  - Population size
  - Number of generations
  - Crossover methods
  - Mutation methods
  - Mutation chance

- After trying a set of options, pick the “best” result from that, and use that as your solution

Random Numbers in Modern C++

C++11 Random

- The original rand() is *implementation-defined*, which means:
  - Not portable
  - May not be thread-safe
  - May not be a great random number generator
  - Code like `rand() % num` isn't a great way to guarantee a specific type of distribution

C++11 Random

- The original rand() is *implementation-defined*, which means:
  - Not portable
  - May not be thread-safe
  - May not be a great random number generator
  - Code like `rand() % num` isn't a great way to guarantee a specific type of distribution

- In C++11 there is now a standardized library in <random>:
  - Defines a set of random number generators with specific implementations
  - Defines specific ways to extract randomized distributions (uniform, Gaussian, etc.)

C++11 Random, Basics

- First, declare an instance of a generator
- We're using the Mersenne Twister PRNG:

```
std::mt19937 randGen(seed);
```

- In our case, the seed value will be a command line parameter. This is to guarantee your results will match my results.

- ***PA Note:*** You must make sure you generate random numbers where the instructions tell you to, or you will get different results.

C++11 Random, Distribution

- First declare the distribution

- Then when you want to use it, you call it like a function, passing in the random number generator:

```
// Pick a random value from [0, 10]
std::uniform_int_distribution<int> myDist(0, 10);
int index1 = myDist(randGen);
```

Distributions

- uniform_int_distribution – Evenly distributed integer range
- uniform_real_distribution – Evenly distributed real range
- binomial_distribution
- normal_distribution – Normal/Gaussian
- Some other stuff too!