PRACTICAL GENETIC ALGORITHMS

Julian Bunn

Center for Data Driven Discovery

Caltech

2015

OVERVIEW

- What are Genetic Algorithms?
- Why use Genetic Algorithms?
- Genetic Algorithm Terminology
- Chromosome Populations and Evolution
- Fitness Functions, Population Mating/Crossover and Mutation
- A Practical Example: the Travelling Salesman Problem

WHAT ARE GENETIC ALGORITHMS?

- GAs are algorithms that transform themselves to solve problems in ways inspired by genetics and natural selection
- They were first described by John Holland at the University of Michigan in the 70s. Holland was influenced by prior work on "evolution strategies" by Rechenberg, Fogel, Owens and Walsh in the 60s.
- Earlier still, Alan Turing had toyed with the idea of evolutionary programs.

- The basic concept:

Evolve a population of randomly generated algorithms so that successive populations improve their ability to achieve a goal.

WHY USE GENETIC ALGORITHMS?

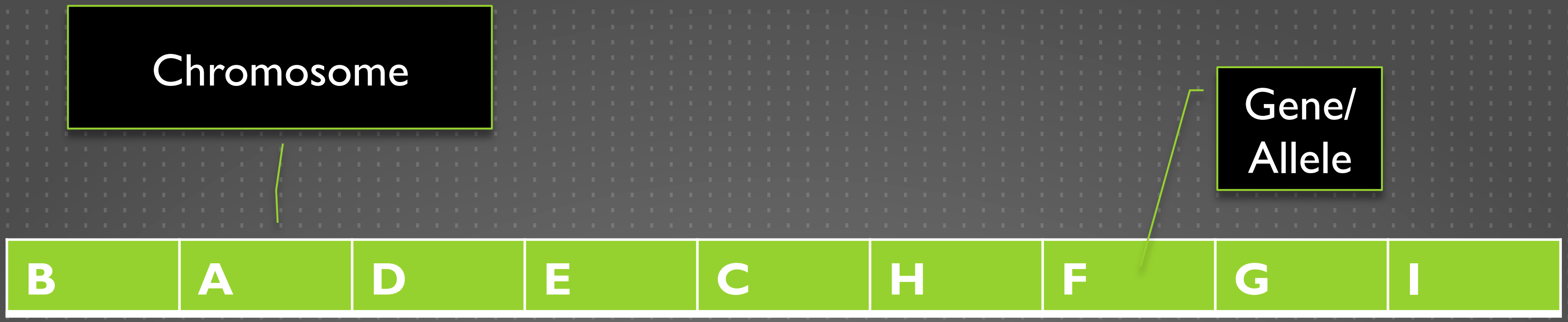
- Good for finding global optimum solutions, and avoiding local optima
- Handy for problems where the variables may not be continuous, but may be discrete, integer, boolean, strings etc.
- Suitable for problems where a non-perfect solution is acceptable
- Effective when a good "fitness" function can be defined (i.e. when a good solution can be specified algorithmically)

- See also: Simulated Annealing, Particle Flow

GA TERMINOLOGY

- Consider Natural Selection:
  - Living organisms battle nature to survive. The fittest organisms (strongest, fastest, biggest, most intelligent) are more likely to survive.
  - The survivors mate and reproduce – their offspring, though different, carry the parents' traits, which they inherit.
  - The offspring may be fitter or less fit than their parents.
  - The population of organisms evolves over generations to have better chances of survival.
- What happens at a biological level:
  - Organisms are individuals specified by a genetic code, which is different for each individual.
  - The genetic code is a chromosome – a long string of genes (instantiated as alleles)
  - The order of the genes, their number and their type, specify the chromosome.
  - When organisms mate, their offsprings' chromosomes are mixtures (crossover) and mutations of the parents.

GA TERMINOLOGY ILLUSTRATED



This chromosome is a list of 9 genes. Each gene is instantiated as a specific character, termed an allele.

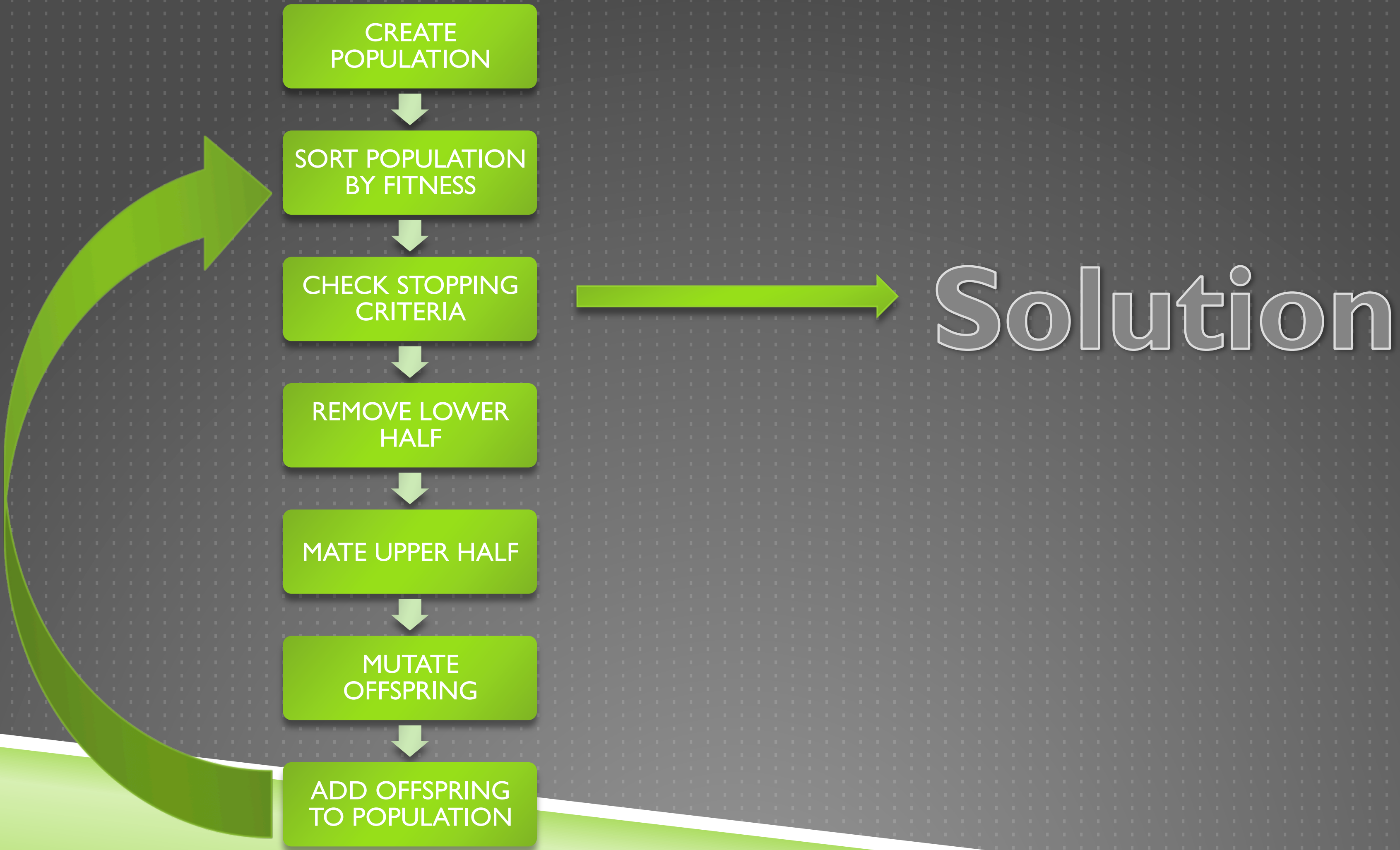
This particular chromosome happens to specify the ordered path a Travelling Salesman might take when travelling between 9 cities.

A POPULATION OF CHROMOSOMES

1	0	1	1	0	0	1	1	1
0	0	0	1	1	1	0	1	0
1	0	0	0	0	0	1	1	0
1	1	1	1	0	0	1	0	1
1	0	1	0	1	0	1	0	1
0	1	1	1	1	0	0	0	1

- A population of 6 chromosomes
- Each gene is a bit – binary encoding
- The chromosomes were generated randomly

GA POPULATION EVOLUTION



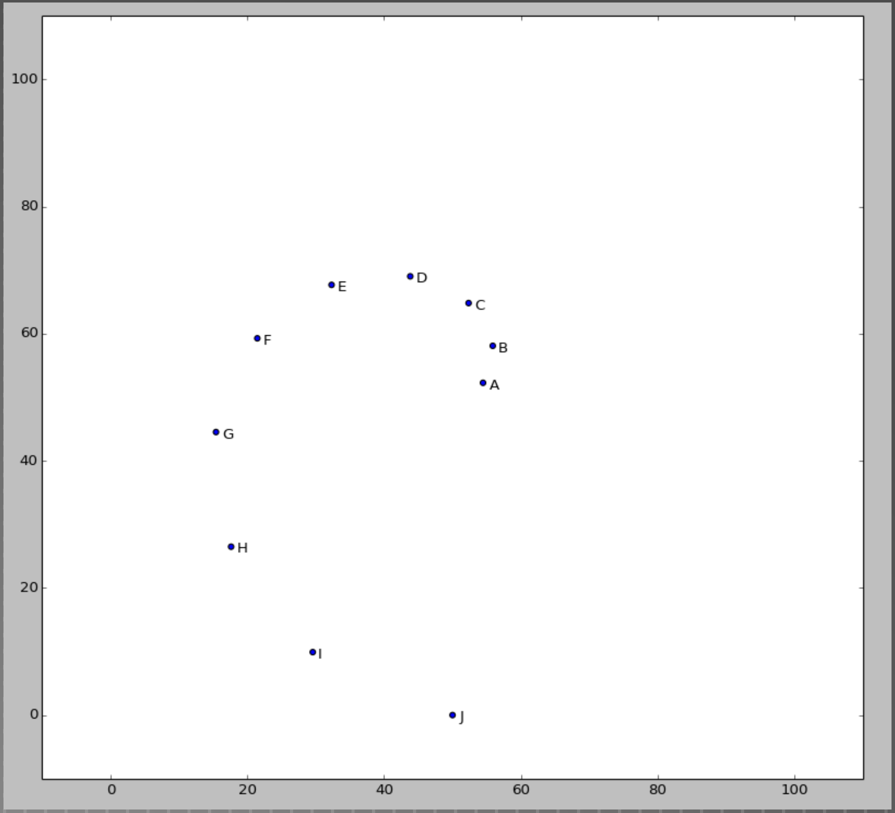
CREATING A GENETIC ALGORITHM

- What is the problem being solved? Example:
  - Find shortest path between cities, visiting all cities only once – Travelling Salesman
- Define the representation of possible solutions to the problem:
  - Travelling Salesman: ordered list of cities
- Define the "fitness" or "cost" function, which measures how well a solution solves the problem.
  - Travelling Salesman: length of the travelled path specified by the ordered city list
- Define the genetic "operators" i.e. how the chromosomes will be initially created, then mate and mutate
- Decide on the "stopping" conditions, i.e. when should the algorithm terminate. Examples:
  - When the best chromosome represents the solution to within an error epsilon
  - When the number of "epochs" (or generations of the population) exceeds a limit
  - When the same chromosome appears as the best in several successive epochs
- Write the code (or use someone else's)

SOLVING A TRAVELLING SALESMAN PROBLEM WITH A GENETIC ALGORITHM

A salesman wants to sell his wares in all the cities in a region. But he is keen to use as little gas/petrol as possible while travelling.

- What is the visit order that minimizes the total distance travelled by the salesman, given:
  - Each city must be visited once and only once
  - The salesman can start anywhere
- The diagram shows 10 hypothetical cities [A-J], arranged on a spiral
- The number of possible paths is 10! = 3,628,800
- It seems intuitive that there are two shortest paths:
  - A-B-C-D-E-F-G-H-I-J, or
  - J-I-H-G-F-E-D-C-B-A



CREATING THE POPULATION

- What should each chromosome look like? Example representation:

B	A	F	H	C	E	D	J	G	I

- To create the population, generate chromosomes with random orderings of the cities.

```
for i in range(NUMBER_OF_CHROMOSOMES):
        city_list = list(cities.keys())
        random.shuffle(city_list)
        chromosomes.append(Chromosome(city_list))
```

"cities.keys()" is the list [A,B,C,D,E,F,G,H,I,J]

"random.shuffle" shuffles the list in place

"Chromosome(city_list)" creates a chromosome with the shuffled city list

DEFINING THE FITNESS FUNCTION

How should the fitness of the chromosome be evaluated?

```
def path_length(city_list, cities):
        sum = 0.0
        for i in range(1,len(city_list)):
            (x1,y1) = cities[city_list[i-1]]
            (x2,y2) = cities[city_list[i]]
            sum += math.sqrt((x1-x2)**2+(y1-y2)**2)
        return sum
```

  - This function calculates the path distance for the ordering in "city_list" given city positions in "cities"
  - The fittest chromosomes will have the smallest values for the function
- In general, deciding on a good fitness function is the main key to the success of working with Genetic Algorithms

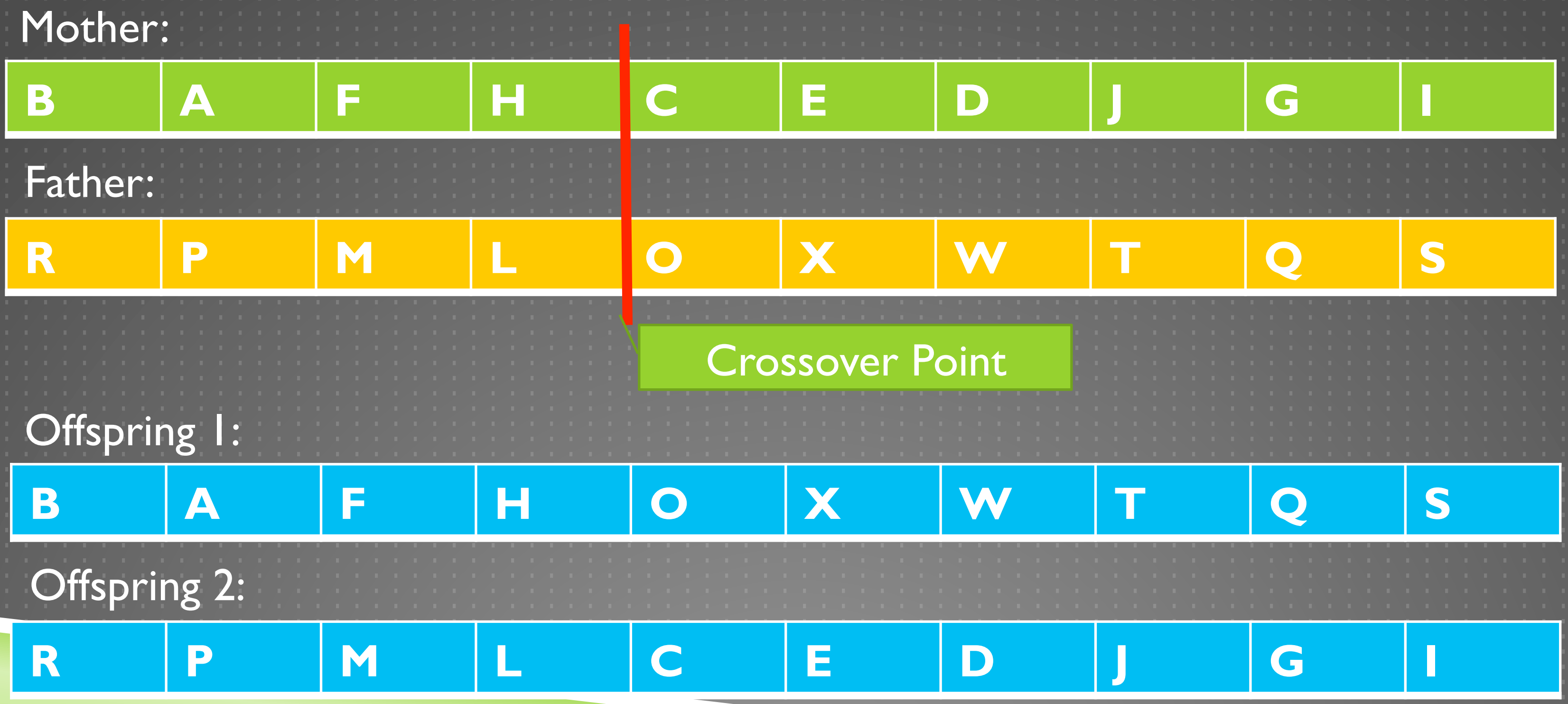
ORDERING AND MATING THE CHROMOSOMES

- Once we have our chromosome population, we order them by fitness.
- Conventionally, we discard the bottom half of the ordered population.
- The top half of the population will be the Mating Population
  - These are the chromosomes which will mate and produce offspring
- Using the Mating Population, we pick pairs of chromosomes at random.
- Each mating pair is used to create a new pair of chromosomes: offspring
  - Offspring may be produced singly, if desired
- Each of the offspring may mutate (i.e. undergo a random change)
- The offspring are added to the Offspring Population
- This continues until there are enough chromosomes in the Mating and Offspring populations combined to match the size of the starting population

CHROMOSOMES MATING: CROSSOVER

Single Point Crossover

- Position in chromosome chosen randomly, first offspring takes alleles from mother up to crossover point, from father beyond. Second offspring vice versa



Mother:

B	A	F	H	C	E	D	J	G	I

Father:

R	P	M	L	O	X	W	T	Q	S

Offspring 1:

B	A	F	H	O	X	W	T	Q	S

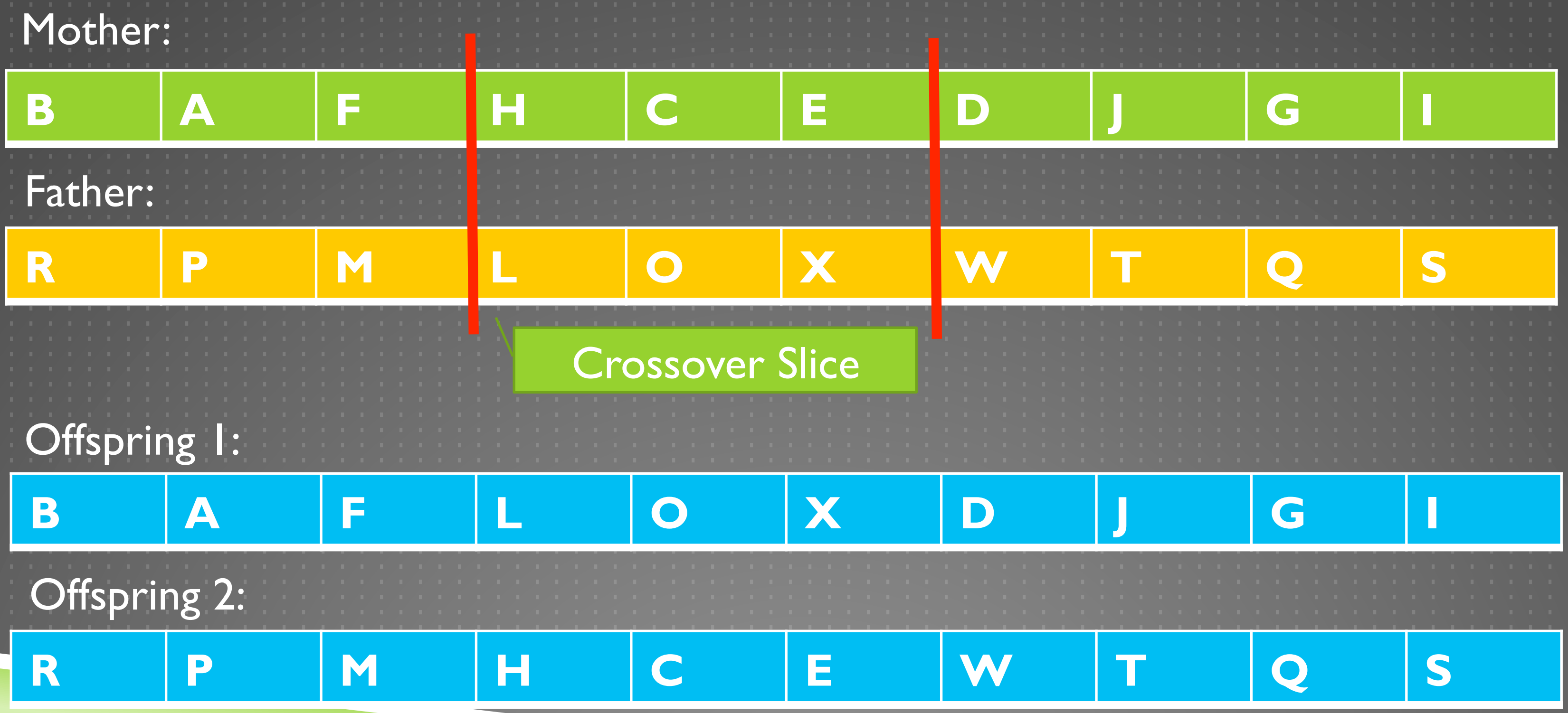
Offspring 2:

R	P	M	L	C	E	D	J	G	I

DOUBLE POINT CROSSOVER

Slice of chromosome chosen randomly, and exchanged.

- Usually a fixed length. First offspring takes the mother's alleles, mixed with the father's slice. Second offspring vice versa.



SPECIAL CROSSOVER METHODS

For problems where the chromosomes must contain unique alleles, a special crossover method is used that ensures no duplication, at the expense of some re-ordering of the offspring chromosomes. Example:

Mother:

3	1	2	4	5	6	8	7	9	0

Father:

2	4	6	8	0	1	3	5	7	9

Preliminary Offspring (with duplicate alleles):

3	1	2	8	0	1	8	7	9	0

Alleles 1,8 and 0 are duplicated. Alleles 4,5 and 6 are missing

Corrected Offspring (no duplicate or missing alleles):

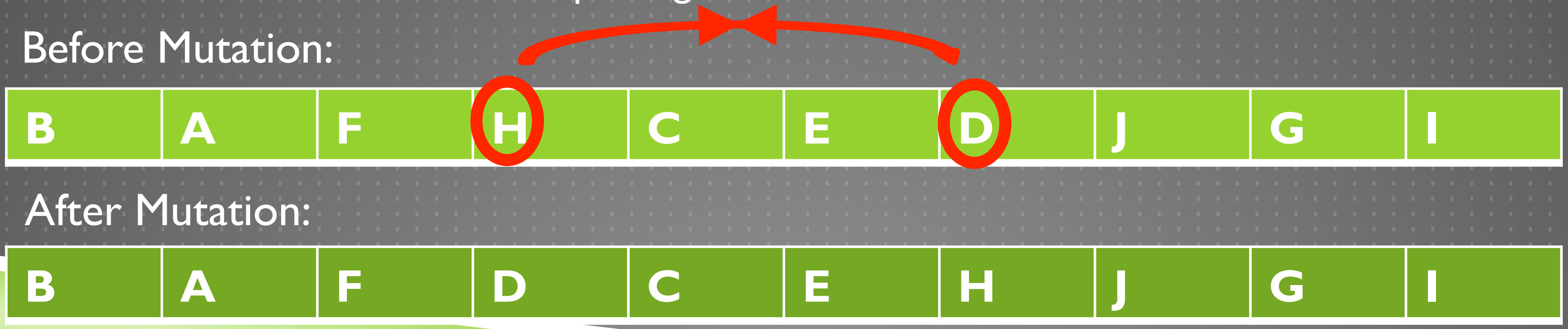
3	1	2	8	0	4	5	7	9	6

MUTATION

- New offspring undergo mutation with some probability.
- This helps to diversify the chromosome population
- Mutation is achieved by applying a random change to the chromosome.

MUTATION

- Typical mutation methods:
  - Pick a random allele in the chromosome and change it to a different, random, value
  - Swap two alleles, chosen randomly, in the chromosome
  - Cut the chromosome at a random position, and place the tail alleles before the head
- Any mutation is valid, as long as it does not violate the semantics of the chromosome representation
  - E.g. changing an allele from a float to a random string will clearly cause problems if the fitness function is expecting all alleles to be floats!



RUNNING THE GENETIC ALGORITHM

- At each epoch (new generation of chromosomes), we order by fitness and decide whether to continue to the next epoch
  - If the best chromosome meets our stopping criteria, we finish
  - If the number of epochs meets a maximum number we specified, we finish
  - If the best chromosome this epoch is identical to the best in several prior epochs, we finish
- The size of the chromosome population, the mutation probability, the crossover size, and the stopping criteria are typically determined heuristically
  - Rule of thumb: chromosome population size > 4 * chromosome length

CODE EXAMPLE – CHROMOSOME MATING

```
def mate(self, father):
        '''
        Mates this mother chromosome with a father and produces a pair of offspring
        '''
        #swap a segment between the mother and the father
        swap_start_position = random.randrange(self.length-self.swap_length)
        swap_end_position = swap_start_position + self.swap_length
        #obtain the list of cut alleles in the father's and mother's chromosomes
        father_cut = father.chromosome[swap_start_position:swap_end_position]
        mother_cut = self.chromosome[swap_start_position:swap_end_position]

        #create a new chromosome with the alleles for the first child
        offspring1 = Chromosome(self.chromosome[:swap_start_position] + \
                                father_cut + \
                                self.chromosome[swap_end_position:])

        # create a new chromosome with the alleles for the second child
        offspring2 = Chromosome(father.chromosome[:swap_start_position] + \
                                mother_cut + \
                                father.chromosome[swap_end_position:])

        return (offspring1, offspring2)
```

CODE EXAMPLE – CROSSOVER WITHOUT ALLELE DUPLICATION (1)

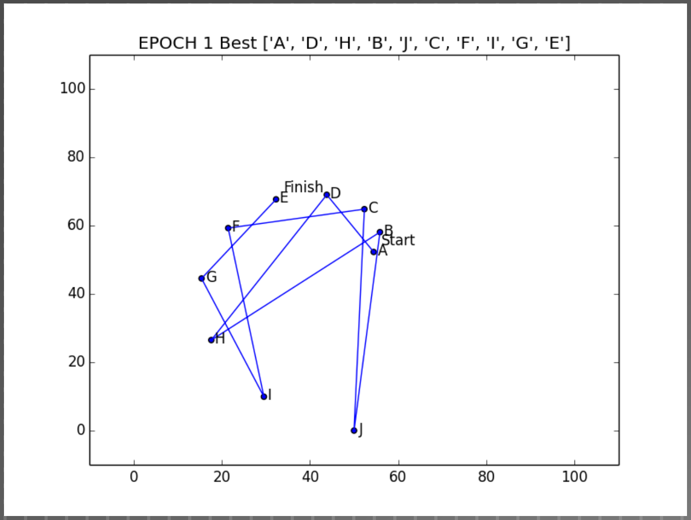
```
  def mate_no_duplicates(self, father):
        '''
        Mates this mother chromosome with a father and produces a pair of offspring.
This mating scheme ensures that the chromosome alleles are not duplicated
        '''
        #swap a segment between the mother and the father
        swap_start_position = random.randrange(self.length-self.swap_length)
        swap_end_position = swap_start_position + self.swap_length
        #obtain the list of cut alleles in the father's and mother's chromosomes
        father_cut = father.chromosome[swap_start_position:swap_end_position]
        mother_cut = self.chromosome[swap_start_position:swap_end_position]

        offspring1_chromosome = self.chromosome[:swap_start_position] + \
                                father_cut + \
                                self.chromosome[swap_end_position:]
        # ensure that we are not missing alleles
        offspring1_chromosome = self.enforce_all_entries(offspring1_chromosome)
        offspring1 = Chromosome(offspring1_chromosome)

        # create a new chromosome with the alleles for the second child
        offspring2_chromosome = father.chromosome[:swap_start_position] + \
                                mother_cut + \
                                father.chromosome[swap_end_position:]
        offspring2_chromosome = self.enforce_all_entries(offspring2_chromosome)
        offspring2 = Chromosome(offspring2_chromosome)

        return (offspring1, offspring2)
```

CODE EXAMPLE – CROSSOVER WITHOUT ALLELE DUPLICATION (2)

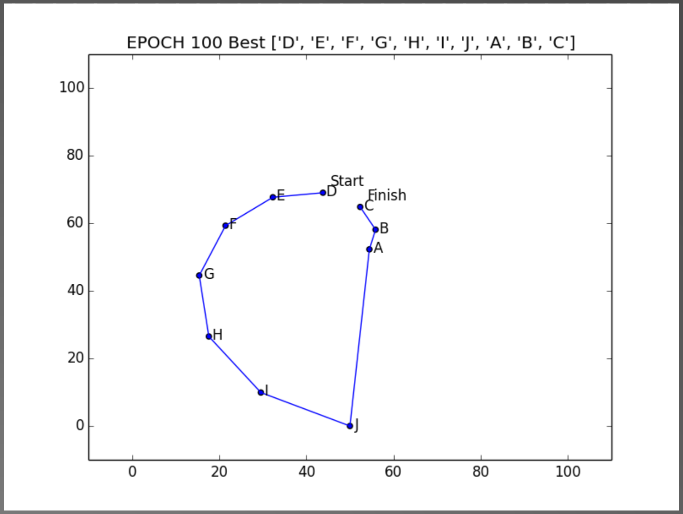
```
def enforce_all_entries(self, target):
  '''
  Ensures that the target list contains all items in the chromosome
  '''
  missing = set(self.chromosome) - set(target)
  while len(missing) > 0:
     duplicates = set([x for x in target if target.count(x) > 1])
     if len(duplicates) == 0:
        break
     target[target.index(duplicates.pop())] = missing.pop()

  return target
```

TRAVELLING SALESMAN EXAMPLE

Epoch 1: the best chromosome in an initial population of 20



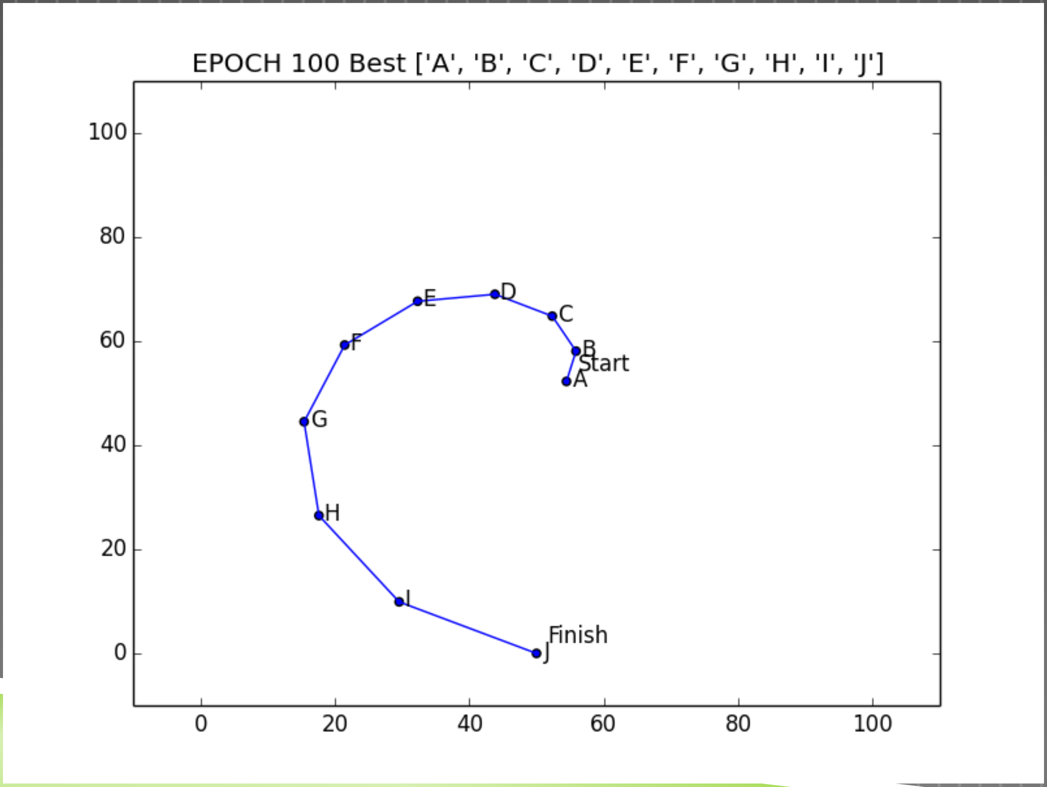
Epoch 100: the best chromosome



The solution is not optimal: we can increase the population size, or run for more epochs….

TRAVELLING SALESMAN SOLUTION

We increased the population size to 100 chromosomes, and obtained the following result after 100 epochs:



MORE ON FITNESS FUNCTIONS

- The most important part of designing a Genetic Algorithm is the creation or identification of a good fitness function
- With a good fitness function, the GA will converge more rapidly on a globally optimal, and desired, solution.
- With a poor choice of fitness function, several things may happen:
  - The GA never converges
  - The GA happily converges to a local optimum
  - The GA converges to a solution to a different problem from intended
  - The GA takes too long to run
  - The GA discards good chromosomes from the population while evolving

EVEN MORE ON FITNESS FUNCTIONS

- In some cases, the fitness function may need to be quite complicated
- If the fitness needs to expressed in terms of a cost, we can express it as e.g.
    - Fitness = -cost, or
    - Fitness = 1/(1+cost)
- … or simply sort the chromosomes by increasing cost at each epoch
- Evaluating the fitness function for each chromosome is an ideal opportunity for parallelization
- Deciding on a good fitness function is harder than it may first appear!

EPILOGUE

- Some applications of GA
  - Timetable optimization and scheduling
  - Electronic circuit reverse engineering and design
  - Code breaking
  - Finding good initial weights for a Neural Network
  - Generating art
- GA toolkits
  - Matlab "Global Optimization Toolbox" http://www.mathworks.com/products/global-optimization/
  - DEAP https://github.com/DEAP/deap
  - PyEvolve http://pyevolve.sourceforge.net/
  - You can download the Python code for the Travelling Salesman example used in this lecture here: http://pcbunn.cacr.caltech.edu/TravellingSalesmanGA.zip
- Explore further
  - Gene Expression Programming – the chromosomes include symbolic expressions
    - http://en.wikipedia.org/wiki/Gene_expression_programming
    - https://github.com/trevorstephens/gplearn#oo (very new)
    - http://en.wikipedia.org/wiki/Symbolic_regression