Systems Biology – Exercises

Week 8: Genetic Algorithm (GA) - Part 1

Genetic Algorithm

- We have covered mechanisms of evolution by natural selection and want to investigate how this idea can be realized by actual computer code. The two mechanisms that we have talked about—and you had time to write as Python code—are *mutation* and *recombination*.
- Mutation: A random change of information from one generation to another.
- Recombination: Also called crossover, gene recombination is another method of changing information from one generation to another. Although the crossover point is randomly determined, it has lesser impact on individual changes and in a way can counteract the gene mutation.

Mutation and Recombination

- Both mechanisms (Mutation and Recombination) will help avoid local optima. However, keep in mind, that this also means the algorithm can escape a global optimum.
- Let's start with setting up conditions and variables for a Genetic Algorithm. This GA will use concepts from evolution to approximate a string (the target DNA). We start with a population size of 20 individuals, all with different random names (DNAs) and "breed" 1000 generations following evolution by natural selection, namely mutating the DNA, recombine the DNA, and select the fittest DNA (all on a generation to generation basis).

GA Conditions and Variables

You should use the string of your real name (in English) to replace the string “Your Name”

```
 6 import random
 7 import string
 8 import numpy as np
 9
10 target = "Your Name"
11 dnaLength = len(target)
12 populationSize = 20
13 generations = 5000
14 mutationChance = 100
```

Code 1. GA Conditions and Variables.

Exercise 1

- **Exercise 1:** Below is a simple function to generate a random ASCII character. Replace the inside of the function with your own code if it is necessary, but keep the name of the function as provided.

```
29 def randomGene():
30     return random.choice(string.printable)
```

Code 2. Create a Random Gene.

Initial Population

- Initial Population: To start off the algorithm, we create an initial population with random DNA.

```
34 def initialPopulation():
35     initPop = []
36     for i in range(populationSize):
37         initPop.append(''.join(random.choice(string.printable) for i in range(dnaLength)))
38     return initPop
```

Code 3. Create the Initial Population.

Fitness Function

- Fitness Function**:** The next function is concerned with the selection process. This will be covered in the next class and for the time being will just return 0.

```
47 def fitnessFunction(competingDNA):
48     fitness = 0
49     return fitness
```

Code 4. Dummy Fitness Function.

Exercise 2

- **Exercise 2:** Include your mutation algorithm that has a 1/100 chance of mutating each gene (not DNA) in the code below. Change your code so it uses the notation below (competingDNA, muatationRatio, mutatedDNA).

```
69 def mutation(competingDNA, mutationRatio):
70     mutatedDNA = ""
71     # Include your algorithm here
72     return mutatedDNA
```

Code 5. Mutation Function.

Exercise 3

- **Exercise 3:** In the function below, include your algorithm that crosses over two DNAs at a random point.

```
86 def recombination(competingDNA1, competingDNA2):
87     # Include your algorithm here
88     return (DNAout1, DNAout2)
```

Code 6. Recombination Function.

Weighted Selection of DNA

- **Weighted Selection of DNA:** To get a working GA, we need several functions that are called each time a new generation is generated.
- Include the following function in your code, we will discuss the meaning and impact from next week.

```
28 def weightedDNAchoice(competingDNAfitnessPairs):
29     probs = [competingDNAfitnessPairs[i][1] for i in range(len(competingDNAfitnessPairs))]
30     probs = np.array(probs)
31     probs /= probs.sum()
32     return competingDNAfitnessPairs[np.random.choice(len(competingDNAfitnessPairs), 1, p = probs)[0]][0]
```

Code 7. Weighted Selection of DNA for Next Generation.

Implementation

- Implementation: Now that we have all necessary functions to write a Genetic Algorithm, let's implement them into the following algorithm (in the next page). The code is roughly outlined with comments. Take some time to understand the flow of the algorithm, add your own comments and replace or extend existing ones!

Code 8

```
currentPopulation = initialPopulation()
for i in range(generations):
    lastfitnessarray = []
    for k in currentPopulation:
        lastfitnessarray.append(fitnessFunction(k))
    # Prints the generation number and its current fittest DNA string
    print("The fittest DNA for generation", i, "is ---", currentPopulation[
            lastfitnessarray.index(min(lastfitnessarray))],
            "--- with penalty:", min(lastfitnessarray))
    # Returns a new population with their respective fitness in format
    # [ ["dnastr1", penalty1], ["dnastr2", penalty2] [...] ... ]
    populationWeighted = []
    for individual in currentPopulation:
        individualPenalty = fitnessFunction(individual)
        if individualPenalty == 0:
            DNAfitnessPair = (individual, 1.0)
        else:
            DNAfitnessPair = (individual, 1.0/individualPenalty)
        populationWeighted.append(DNAfitnessPair)

    # Reset population and repopulate with newly selected, recombined, and mutated DNA
    currentPopulation = []
    for m in range(int(populationSize/2)):
        # Random selecion, weighted by fitness (higher fitness == higher probability)
        fittestDNA1 = weightedDNAchoice(populationWeighted)
        fittestDNA2 = weightedDNAchoice(populationWeighted)
        # Recombination or crossover
        fittestDNA1, fittestDNA2 = recombination(fittestDNA1, fittestDNA2)
        # Mutation in 1/mutationChance chances
        fittestDNA1 = mutation(fittestDNA1, mutationChance)
        fittestDNA2 = mutation(fittestDNA2, mutationChance)
        # Combining the population for next iteration
        currentPopulation.append(fittestDNA1)
        currentPopulation.append(fittestDNA2)
# Creates an array of penalty value for each DNA in population
lastfitnessarray = []
for g in currentPopulation:
    lastfitnessarray.append(fitnessFunction(g))
# Prints fittest DNA out of the resulting population
print("Fittest String at", generations, "is:",
      currentPopulation[lastfitnessarray.index(min(lastfitnessarray))])
```

Homework

- **Homework:** Due next Wednesday (17:00, Nov. 24th, 2021) electronically to manaba+R.
- File format: YourStudentID_Week08.py (ID without hyphen, e.g., 12345678901_Week08.py).
- Your code must include your own comments for all code sections. Go line-by-line. Comments in your program must be full sentences and reflect your understanding of the code.
- Q1. Create one Python file, which includes all necessary functions and the implementation to run the Genetic Algorithm (without changing the fitness function). The output will result in gibberish, but already includes processes such as mutation and recombination (please note that the example codes 5 and 6 do not include the processes of mutation and recombination).

Systems Biology – Exercises

Week 9: Genetic Algorithm (GA) - Part 2