CS-415: Computational Biology: Project 1

Taylor Martin
University of Idaho
Moscow, Idaho, USA
mart8517@vandals.uidaho.edu

ACM Reference Format:
Taylor Martin. 2023. CS-415: Computational Biology: Project 1. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 ABSTRACT

Currently in CS-415, we have developed a simple genetic algorithm framework to:

(1) Generate individuals.
(2) Give the individuals a randomly generated genome.
(3) Measure the individuals fitness.
(4) Mutate the individuals genome.
(5) Insert individuals in populations which consist of numerous individuals.
(6) Calculate the average fitness of a population of individuals.
(7) Pick the most fit individuals from the population for reproduction.

I used this framework to perform two experiments:

(1) Increasing the Individual Mutation Rate With Populations Of Increasing Size Magnitude.
(2) Improving The Calculate Fitness Function And Seeing Its Effects On Various Populations Average Fitness.

The results of the first experiment showed that increasing the mutation rate of individuals while increasing the population size does not always lead to a higher rate of mutated individuals in a population. Nor does it always lead to a population with a overall higher fitness score.

The results of the second experiment showed that creating an improved algorithm for calculating an individuals fitness, that is more strict and biologically accurate, will lead to populations that have decreased average fitness scores overall.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 EXPERIMENT 1: INCREASING THE INDIVIDUAL MUTATION RATE WITH POPULATIONS OF INCREASING SIZE MAGNITUDE

For my first experiment, I compared the effects of increasing the individual mutation rate on populations of various size magnitudes. My Hypothesis: Increasing population sizes of individuals with higher mutation rates would lead to a majority of mutated individuals with a higher fitness score.

2.1 Experiment 1a: Individual Mutation Rate Of 5% With Populations Of Increasing Size Magnitudes

For the first half of Experiment 1, I set the individual mutation rate to 5%. I then created a list of 10 populations called populations_1. Starting with an initial population size of 50 individuals and increasing each population size by a magnitude of 50 until we had a max population of 500. I did this by utilizing a function I wrote called create_population_of_increasing_magnitude(). This returned a list of 10 populations, sizes of: 50,100,150,200,250,300,350,400,450,500.

```
populations_1 =
     create_populations_of_increasing_magnitude(50, 10,
     50)
```

I then utilized a function that I wrote called mutate_population_lists() to mutate the populations with an individual mutation rate of 5% that I have contained within my population_1 list.

```
mutate_population_lists(populations_1, 5)
```

Once I had my list of 10 populations and the populations had all been mutated, I created a new list called mean_population_fitnesses_1. To contain the average fitness of each of the populations contained within populations_1.

```
mean_population_fitnesses1 = []

for index in range(0, len(populations_1)):
   mean_population_fitnesses1.append(populations_1[index].avg_fitness)
```

I plotted the mean population fitness' from the mean_population_fitnesses_2 list on a scatter plot using the matplotlib python library.

```
import matplotlib.pyplot as plt
```

```
    plt.scatter([50, 100, 150, 200, 250, 300, 350, 400,
         450, 500], mean_population_fitnesses1)
    plt.xlabel('Population Size')
    plt.ylabel('Average Fitness')
    plt.title('Average Fitness vs Population Size with 5%
         Mutation Rate')
    plt.show()
```

I then created a dictionary, to hold the lists of the average fitness, for each of the populations contained within a populations list.

```
    means_of_populations = {'means_of_population1' :
         mean_population_fitnesses1}
```

2.2 Experiment 1b: Individual Mutation Rate Of 25% With Populations Of Increasing Size Magnitudes

For the second half of Experiment 1, I set the individual mutation rate to 25%. I then created a new list of 10 populations called populations_2. Starting with an initial population size of 50 individuals and increasing each population size by a magnitude of 50 until we had a max population of 500. I did this by again utilizing a function I wrote called create_population_of_increasing_magnitude(). This returned a list of 10 populations, sizes of: 50,100,150,200,250,300,350,400,450,500.

```
populations_2 =
     create_populations_of_increasing_magnitude(50, 10, 50)
```

Once I had my new list of populations, I again utilized the mutate_population_lists(). To mutate the populations with an individual mutation rate of 25% that I have contained within my population_2 list.

```
mutate_population_lists(populations_2, 25)
```

Once I had my new list of 10 populations and the populations had all been mutated, I created another new list called mean_population_fitnesses_2. To contain the average fitness of each of the populations in populations_2.

```
    mean_population_fitnesses1 = []

    for index in range(0, len(populations_1)):
       mean_population_fitnesses1.append(populations_1[index].avg_fitness)
```

I plotted the mean population fitness' from the mean_population_fitnesses_2 list on a scatter plot using the matplotlib python library.

```
    plt.scatter([50, 100, 150, 200, 250, 300, 350, 400,
         450, 500], mean_population_fitnesses2)
    plt.xlabel('Population Size')
    plt.ylabel('Average Fitness')
    plt.title('Average Fitness vs Population Size with 25%
         Mutation Rate')
    plt.show()
```

I updated the means_of_populations dictionary, to hold the new list mean_population_fitnesses_2.

```
    means_of_populations.update({'means_of_population2' :
         mean_population_fitnesses2})
```

After both halves of Experiment 1 were completed, I plotted the mean population fitness' from the means_of_populations dictionary on a scatter plot using the matplotlib python library to compare the data.

```
    plt.scatter([50, 100, 150, 200, 250, 300, 350, 400,
         450, 500],
         means_of_populations['means_of_population1'],
         label = '5% Mutation Rate', color = 'red')
    plt.scatter([50, 100, 150, 200, 250, 300, 350, 400,
         450, 500],
         means_of_populations['means_of_population2'],
         label = '25% Mutation Rate', color = 'blue')
    plt.xlabel('Population Size')
    plt.ylabel('Average Fitness')
    plt.title('Average Fitness vs Population Size with
         Various Mutation Rates')
    plt.legend()
    plt.show()
```

3 EXPERIMENT 2: IMPROVING THE CALCULATE FITNESS FUNCTION AND SEEING ITS EFFECTS ON VARIOUS POPULATIONS AVERAGE FITNESS

The next experiment that I performed was comparing different fitness calculation algorithms. I first used the simpler calculate_fitness_function() that we created in lecture. I then constructed a slightly more strict and biologically accurate fitness algorithm that will calculate the number of triplet "T" occurrences within an individuals genome. Without replacement, so it will not count duplicate triplets. I will then see how this affects the average fitness of populations of various sizes. My hypothesis: creating an improved algorithm for calculating an individuals fitness, that is more strict and biologically accurate, will lead to populations that have decreased average fitness scores overall.

3.1 Experiment 2a: Using Original Calculate Fitness Function And Seeing Its Effects On Average Fitness For Populations Of Increasing Size Magnitudes

For the first half of Experiment 2, I created a list of 10 populations called populations_1. Starting with an initial population size of 50 individuals and increasing each population size by a magnitude of 50 until we had a max population of 500. I did this by utilizing

create_population_of_increasing_magnitude(). This returned a list of 10 populations, sizes of: 50,100,150,200,250,300,350,400,450,500.

```
populations_1 =
    create_populations_of_increasing_magnitude(50, 10,
    50)
```

When a population is first initialized with instances of individuals, the individual class constructor calls a calculate_fitness() function to calculate the fitness of each individual in a population. The constructor then calls a calculate_population_stats function to calculate the average fitness of a population of individuals.

Once I had my list of 10 populations and the populations fitness' had all been calculated with the old calculate_fitness() function. I created a new list called mean_population_fitnesses_1. To contain the average fitness of each of the populations contained within populations_1.

```
mean_population_fitnesses1 = []

for index in range(0, len(populations_1)):
   mean_population_fitnesses1.append(populations_1[index].avg_fitness)
```

I plotted the mean population fitness' from the mean_population_fitnesses_2 list on a scatter plot using the matplotlib python library.

```
plt.scatter([50, 100, 150, 200, 250, 300, 350, 400, 450,
    500], mean_population_fitnesses1)
plt.xlabel('Population Size')
plt.ylabel('Average Fitness')
plt.title('Average Fitness vs Population Size with
    Original Fitness Function')
plt.show()
```

I then created a dictionary, to hold the lists of the average fitness, for each of the populations contained within a populations list.

```
means_of_populations = {'means_of_population1' :
    mean_population_fitnesses1}
```

3.2 Experiment 2b: Using Improved Calculate Fitness Function And Seeing Its Effects On Average Fitness For Populations Of Increasing Size Magnitudes

For the second half of Experiment 2, I created a list of 10 populations called populations_2. Starting with an initial population size of 50 individuals and increasing each population size by a magnitude of 50 until we had a max population of 500. I did this by utilizing create_population_of_increasing_magnitude(). This returned a list of 10 populations, sizes of: 50,100,150,200,250,300,350,400,450,500.

```
populations_2 =
    create_populations_of_increasing_magnitude(50, 10,
    50)
```

When a population is first initialized with instances of individuals, the individual class constructor now calls a improved_calculate_fitness() function to calculate the fitness of each individual in a population. The constructor then calls a calculate_population_stats function to calculate the average fitness of a population of individuals. .

Once I had my list of 10 populations and the populations fitness' had all been calculated with the improved_calculate_fitness() function. I created a new list called mean_population_fitnesses_2. To contain the average fitness of each of the populations contained within populations_2.

```
mean_population_fitnesses2 = []

for index in range(0, len(populations_1)):
   mean_population_fitnesses2.append(populations_1[index].avg_fitness)
```

I plotted the mean population fitness' from the mean_population_fitnesses_2 list on a scatter plot using the matplotlib python library.

```
plt.scatter([50, 100, 150, 200, 250, 300, 350, 400, 450,
    500], mean_population_fitnesses2)
plt.xlabel('Population Size')
plt.ylabel('Average Fitness')
plt.title('Average Fitness vs Population Size with
    Strict Fitness Function')
plt.show()
```

I updated the means_of_populations dictionary, to hold the new list mean_population_fitnesses_2.

```
means_of_populations.update({'means_of_population2' :
    mean_population_fitnesses2})
```

After both halves of Experiment 2 were completed, I plotted the mean population fitness' from the means_of_populations dictionary on a scatter plot using the matplotlib python library to compare the data.

```
plt.scatter([50, 100, 150, 200, 250, 300, 350, 400,
    450, 500],
    means_of_populations['means_of_population1'],
    label = 'Original Fitness Function', color = 'red')
plt.scatter([50, 100, 150, 200, 250, 300, 350, 400,
    450, 500],
    means_of_populations['means_of_population2'],
    label = 'Strict Fitness Function', color = 'blue')
plt.xlabel('Population Size')
plt.ylabel('Average Fitness')
plt.title('Average Fitness vs Population Size with
    Various Fitness Functions')
plt.legend()
plt.show()
```

4 ALGORITHMS

Here I will show and explain the code for the algorithms that I wrote in addition to the ones we wrote in lecture in order to complete my experiments.

4.1 create_populations_of_increasing_magnitude():

This function accepts three integer parameters: an initial population size, number of populations, and magnitude of increase. It will initialize an empty list of populations. Then, in a loop, it will add a new population to the list based on the initial population size. Increasing the population size to be created by the magnitude of increase. It will terminate once the list contains the number of populations wanted. returning the list.

```
def create_populations_of_increasing_magnitude(
initial_population_size,
number_of_populations,
magnitude_of_population_increase):
   """create a population of a set magnitude"""
   population_list = []

   for i in range(0,number_of_populations):
      population_list.append(population(initial_population_size))
      initial_population_size +=
           magnitude_of_population_increase

   return population_list
```

4.2 mutate_population_lists():

This function accepts two integer parameters: list of populations to mutate, rate of individual mutation. The function will loop through the sent in list and mutate each of the populations contained.

```
def mutate_population_lists(population_list, mutation_rate):
   """mutate a list of populations"""
   for population in population_list:
      population.mutate_population(mutation_rate)
```

4.3 calculate_fitness():

This function is used by the individual class constructor. It will calculate the fitness of an individual by counting the number of 'T' character occurrences within the individuals genome.

```
def calculate_fitness(self):
  #reset the fitness of the individual to 0
  self.fitness = 0

  #go through the genome and count the number of 'T'
       characters
  for c in self.genome:
     if c == 'T':
        self.fitness += 1
```

4.4 improved_calculate_fitness():

This function is used by the individual class constructor. It will calculate the fitness of an individual by counting the number of 'T' character triplets, without replacement, within the individuals genome.

```
def improved_calculate_fitness(self):

  #reset the fitness of the individual to 0
  self.strict_fitness = 0

  #set the index to 0
  i = 0

  #go through the genome and count the number of 'T'
       character triplets
  while i < genome_size - 2:
     #if the current character is a 'T' and the next two
          characters are 'T'
     if self.genome[i] == 'T' and self.genome[i + 1] == 'T'
          and self.genome[i + 2] == 'T':
        #add 1 to the fitness
        self.strict_fitness += 1
        #increment the index by 3
        i += 2
     #else no 'T' character triplets were found
     #increment the index by 1
     i += 1
```

4.5 calculate_population_stats():

This function is used by the individual class constructor. It will calculate the average fitness of the population by adding up the fitness of the individuals within the population and then dividing the average fitness score by the number of individuals in the population.

```
def calculate_population_stats(self):
  self.avg_strict_fitness = 0

  for i in self.the_population:
     self.avg_strict_fitness += i.strict_fitness

  self.avg_strict_fitness /= self.get_population_size()
```

5 RESULTS

Listed below are the scatter plot results generated from the experiments that I performed:

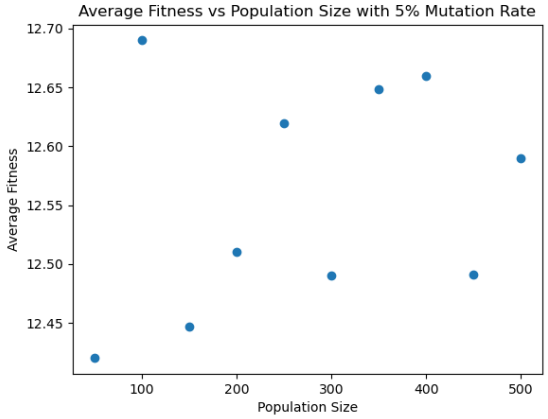

Figure 1: Average Fitness vs Population Size with 5% Mutation Rate

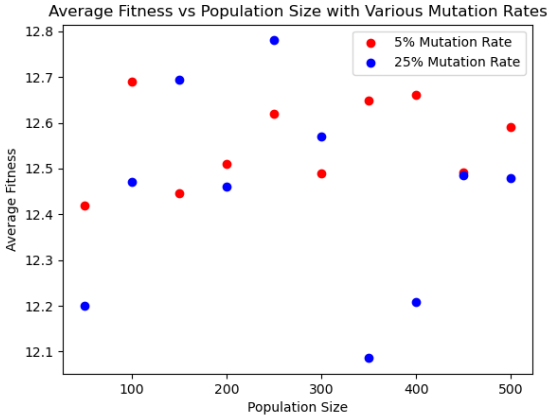

Figure 3: Average Fitness vs Population Size with Various Mutation Rates

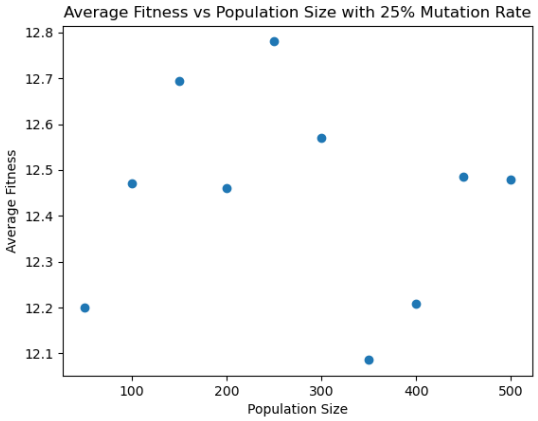

Figure 2: Average Fitness vs Population Size with 25% Mutation Rate

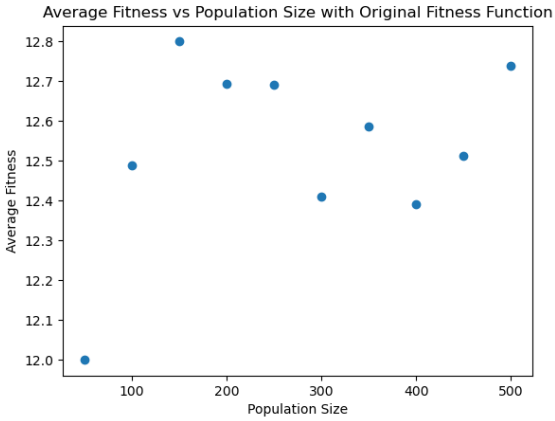

Figure 4: Average Fitness vs Population Size with Original Fitness Function

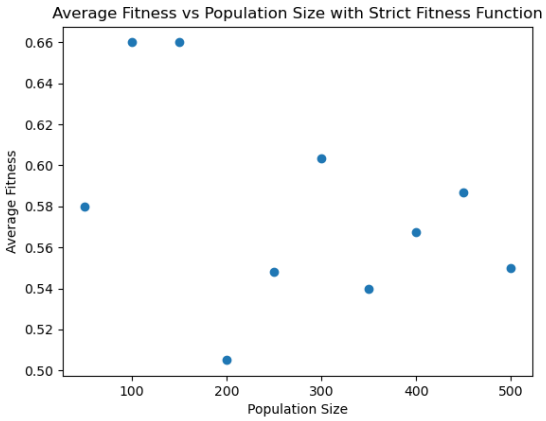

Figure 5: Average Fitness vs Population Size with Improved Fitness Function

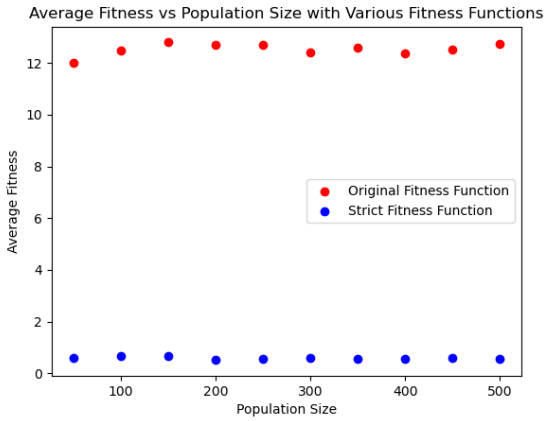

Figure 6: Average Fitness vs Population Size with Various Fitness Functions

6 CONCLUSION

The results of the first experiment showed that increasing the mutation rate of individuals while increasing the population size does not always lead to a higher rate of mutated individuals in a population. Nor does it always lead to a population with a overall higher fitness score.

The results of the second experiment showed that creating an improved algorithm for calculating an individuals fitness, that is more strict and biologically accurate, will lead to populations that have decreased average fitness scores overall.