CS-415/515

Bryce Hendrickson, Molly Meadows, Carson Sloan

Sub Project 2b

3-18-2023

**Genetic Algorithm:**

The given genetic algorithm uses two classes, ‘individual’ and ‘population’, together these classes can initialize and start to evolve a population using a generational method. Below are the algorithm descriptions for the classes ‘individual’ and ‘population’.

Genome characters: ‘TGCA’

Class individual contains four methods:

· Initialization: Initializes a new individual.

1. The individual is assigned a blank ‘fitness’ score of 0.

2. The individual is assigned a blank ‘genome’ character array.

3. The genome array is then generated: the array is filled with a default character of ‘T’ for the entirety of the genome size.

§ Size 192 is used for a 3-character permutation. (4^3 \* 3)

4. The individual then calls the ‘calcFitness’ function which assigns the individuals fitness.

· ‘calcFitness(self)’: Calculates the fitness of the individual. Fitness is determined by the ‘diversity’ of the genome’s codon length, which is defined by the number of distinct permutations of the given codon length. For this experiment we will use a 3-character codon length.

1. The individual’s fitness is assigned a score of 0.

2. A 2D matrix named ‘permutations’ is declared.

3. Runs a loop from 0 to the genome size, running in steps of 3:

§ A string named ‘current’ is assigned 3 consecutive characters from the genome starting from the step.

§ If ‘current’ is not in the ‘permutations’ matrix then the fitness score is increased by one and ‘current’ is added into ‘permutations’ for future checking

4. ‘permutations’ is reset to a blank 2d matrix to avoid memory leakage.

· ‘mutation(self)’: Mutates the individual’s genome. The mutation rate used for this experiment is 2 %.

1. Runs a loop from 0 to the genome size, running in steps of 3.

§ A random number is generated between 0-99 using pythons random.uniform method.

§ If that number is less than our tested mutation rate (2%) then the 3 length codon is randomly assigned a new character for each character in the codon. The experiment uses pythons random.choice(‘TACG’) to choose a new character.

2. The individual then calls the ‘calcFitness’ method to determine the new fitness.

· ‘copy(self,source)’: Copies the source individual’s internal information into the self individual.

1. Assigns the fitness of the self to the source’s fitness.

2. A loop runs from 0 to the genome size

§ Every genome position in self is assigned the source genome’s value.

Class Population contains four methods:

· Initialization: Initializes a new population

1. Assigns the average fitness to 0.

2. Declares the population array.

3. A loop runs from 0 to the population size (100 - static):

§ Appends a new individual onto the population array.

· ‘calcstats(self)’: Calculates the average fitness for the population.

1. The population’s average fitness is assigned to 0.

2. A loop runs for every individual in the population:

§ The population’s average fitness is increased by the individual’s fitness.

3. The population’s average fitness is divided by the population size (100 – static)

· ‘tournament(self)’: Tournament selection against the population, returns the index of the individual who won. (Tournament size is 3 – static)

1. Assigns the ‘bestIndividual’ to a random individual of the population. Using pythons random.randint method.

2. Assigns the ‘bestFitness’ to the fitness of the ‘bestIndividual’.

3. Runs a loop from 0 to the tournament size -1:

§ Assigns a ‘current’ individual to a random individual of the population (as above).

§ Assigns the ‘currentFitness’ to the firness of the ‘current’ individual.

§ If the ‘currentFitness’ is greater than the ‘bestFitness’ then the ‘bestIndividual’ is assigned to the ‘current’ individual and the ‘bestFitness’ is assigned to the ‘currentFitness’.

4. Returns the index of the ‘bestIndividual’

· ‘generational(self)’: simulates an evolutionary generation for the population.

1. Creates a temporary population.

2. Runs a loop from 0 to the population size:

§ Grabs an index of a parent from the ‘tournament’ method

§ Copies the parent individual into the temporary population.

§ Calls the ‘mutation’ method on the temporary population’s new parent individual.

3. Runs a loop from 0 to the population size:

§ Copies each individual from the temporary population into the ‘self’ population.

4. Calls the ‘calcstats’ method on the population to get the new avg fitness.

**Data Description:**

**40 genomes per population (2 populations)**

**Genome length: 192**

The data was generated by running two distinct populations through 100 generations. Every 20th generation 10 individuals from each population were written into their respective files for population one and two.