Cairo University
Faculty of Computer and Artificial Intelligence

Computational Intelligence

DS313 / DS351

Genetic Algorithm

Dr. Ayman Sabry Ghoneim

By

Omar Yousef Mohammed

ID

20210591

Introduction:

- A Python program that implements a genetic algorithm to solve an optimization problem with second degree.

$$\text{Maximize } F(x_1, x_2) = 8 - (x_1 + 0.0317)^2 + (x_2)^2, \text{ where } -2 \leq x_1, x_2 \leq 2.$$

- X1, X2 are in range (-2,2).
- Using the binary representation of solutions.
- Roulette Wheel technique used for selection operation.
- One-point crossover used to do crossover operation.
- Mutation operation is done using bit-flip mutation.
- Population size is 100, chromosome length is 10 bits, each variable takes 5 bits to represent its integer value and the number of generations is 100.

Algorithm Steps:

1- First, Initialize the population randomly with binary numbers (0,1).

2- Calculate the fitness of all solutions by decoding the solutions into integer representation and substituting the values in the objective function. We use two types of decoding standard decoding and gray code decoding.

3- Compute the fitness probabilities of each solution.

4- Select the individuals based on these probabilities as two parents to mix them to introduce new individuals as children in the new generation.

5- Apply Elitism.

6- Perform mutation in the new generation.

7- Return the final generation and best fitness over generations as output.

Population initializing:

- In this implementation, I select the initial population randomly with 100 chromosomes and a chromosome length is 10.

```python

Pop = np.random.randint(0,2 ,size=(20,10))# Initial population with random values


```

Fitness Computation:

- We Should decode the solutions from binary into integer values to calculate the fitness of a solution.
- Standard decoding is the way to decode the solutions and we use gray code decoding as well to prevent hamming cliff problem.
- Standard decoding function the program splits the solution into two parts and passes them to the decode function and it returns integer values.

```python
# This function decodes it into integers and return two integer numbers
def binaryToInteger(part1, part2):
    # Two parts with the same length
    x_min = -2
    x_max = 2
    sum1 = 0
    sum2 = 0
    for i in range(len(part1)):
        sum1 += part1[i] * (2 ** (len(part1) - i - 1))
        sum2 += part2[i] * (2 ** (len(part2) - i - 1))
    x1 = round(x_min + (sum1 / (2 ** len(part1) - 1)) * (x_max - x_min), 4)
    x2 = round(x_min + (sum2 / (2 ** len(part2) - 1)) * (x_max - x_min), 4)
    return x1, x2
```

- Gray code decoding function, I convert gray code into binary and from binary to integer as the above function.

```python
def graycodeToBinary(part1 , part2):
    # Two parts are the same length
    length = len(part1)
    binary1 = np.zeros(length , dtype=int)
    binary1[0] = part1[0]
    binary2 = np.zeros(length,dtype=int)
    binary2[0] = part2[0]
    for i in range(1,length):
        binary1[i] = binary1[i-1] ^ part1[i] #XOR Operation to convert gray into binary
        binary2[i] = binary2[i-1] ^ part2[i]
    return binary1, binary2
```

- After decoding, pass the integer values to the objective function to compute the fitness of the solution.
- After computing the fitness of all solutions, the next step is to select two parents based on this fitness.

Selection:

- Selection using normal roulette wheel.

```python
# Selection using roulette wheel technique
def selection(pop, Fitness):
    # This prevents the probability of the total fitness be equal zero
    probability = adjustedProbabilities(Fitness)
    # Compute the cumulative probability of the fitness
    cumulative = np.cumsum(probability)
    i = 0
    j = 0
    # Get two distinct parents based on their probabilities
    while i == j:
        i = 0
        j = 0
        num1 = random.random()
        num2 = random.random()
        while num1 > cumulative[i]:
            i += 1
        while num2 > cumulative[j]:
            j += 1
    return pop[i] , pop[j]
```

Crossover:

- The recombination technique used in this algorithm is one-point crossover. Random cut index and merge.

```python
# Recombination operation using one-point crossover
def crossover(p1, p2, pCross):
    num = random.random()
    if num < pCross:
        # Take the index of separation randomly
        split_point = random.randint(1, len(p1) - 1)
        firstChild = np.concatenate((p1[:split_point] , p2[split_point:]))
        secondChild = np.concatenate((p2[:split_point] , p1[split_point:]))
        return firstChild, secondChild
    else:
        return p1, p2
```

Mutation:

- As the same in the previous algorithm the bit flip mutation is used as well with random selection of the bit to be flipped.

```python
# Mutation Operation using bit-flip mutation way
def mutation(pop, pMutation):
    for i in range(len(pop)):
        num = random.random()
        if num < pMutation:  # mutate this solution
            randomIndex = random.randint(0, len(pop[0]) - 1)
            if pop[i][randomIndex] == 0:
                pop[i][randomIndex] = 1
            else:
                pop[i][randomIndex] = 0
        else:  # No Mutation, then continue
            continue
```