Genetic-Algoirthm

June 18, 2021

[155]:
```
import random as rand
import numpy.random as rand_2
```

[156]:
```
## Convert N DIM  X N Dim Array into 1 X N Array Dim
```

[157]:
```
# Problem :     [  [0, 0, 0, 0],
#                  [0, 0, 0, 0],
#                  [0, 0, 0, 0],
#                  [0, 0, 0, 0]  ]
# Goal:    [  [1, 1, 1, 1],
#             [1, 1, 1, 1],
#             [1, 1, 1, 1],
#             [1, 1, 1, 1]  ]

# Generating population randomly
def generate_population(array):
    temp_array = array.copy()
    chromosome = []

    for i in range(len(array)):  # Iterating over 2D array
        row = array[i]    # Taking row or element of 2D array
        chromosome.extend(row)   # As chromosome is 1D array

    population = [chromosome.copy() for _  in range(4)]   # Initializing 4␣
↪chromosmes for populaiton

    for i in range(4):
        print("---> ",population[i])
        individual_chromosome = population[i]

        for rand_ind in [rand.randrange(0, len(array)) for _ in range(8)]:   #␣
↪Generating random indices to change value of
            individual_chromosome[rand_ind] = 1

    return population

# calculating fitness value of any individual
```

```
def calculate_fitness(individual):
    num_of_ones = 0

    for i in range(len(individual)):
        num_of_ones = num_of_ones + individual[i]    # As each value is simply 0␣
↪or 1 so we can know number of 1's by siply adding

    return num_of_ones

# Check if any individual is fit enough to be the goal state or desirable
def individual_fit_enough(population, fitness_threshold=16):
    fittest_individual = max(population, key=calculate_fitness)

    if calculate_fitness(fittest_individual) == fitness_threshold:  # If␣
↪fitness value of any individual is equal to 16
        return True

    return False

# Randomly select any individual from population by its selection_prob
def select(population, calculate_fitness):
    fitness_values = [calculate_fitness(individual) for individual in␣
↪population]  # fitness values of population
    sum_fitness = sum(fitness_values)  # sum of fitness values
    selection_probs = [fitness_values[index]/sum_fitness for index, individual␣
↪in enumerate(population)]  # Array holding selection prob for each␣
↪individual  selection_prob(of individual) = fitness_value(of individual) /␣
↪sum_of_all_fitness_values

    population_indices = [ind for ind in range(len(population))]    # Indices of␣
↪individual in population, necessary for numpy.random.choice as it requires␣
↪a=1D array
    selected_individual_index = rand_2.choice(population_indices,␣
↪p=selection_probs)   # p is the selected_probs of corresponding individuals

    return population[selected_individual_index]

# Reproduce a child using x and y chromosme
def reproduce(x, y):
    n = len(x)
    c = rand.randint(0,n)
    return x[:c] + y[c:]  # x upto random n index extended with y from random n␣
↪index

# mutate a chile using prob 0.1
def mutate(child):
```

```
    new_child = child.copy()
    rand_index = rand.randint(0, len(child)-1)  # len(child)-1 as we need␣
↪between 0 to 15 index value
    rand_value = rand.randint(0, 1)
    new_child[rand_index] = rand_value
    return new_child


# genetic search
def genetic_search(population, calculate_fitness):

    while not individual_fit_enough(population):   # repeat until any␣
↪individual is fit enough or time limit
        new_population = []

        for i in range(len(population)):
            x = select(population, calculate_fitness)
            y = select(population, calculate_fitness)

            child = reproduce(x, y)
            if (rand.uniform(0,1) <= 0.1):  # mutate with prob 0.1
                child = mutate(child)

            new_population.append(child)

        population = new_population

    return max(population, key=calculate_fitness)   # return most fitted
```

[158]:
```
population = generate_population([  [0, 0, 0, 0],
                                    [0, 0, 0, 0],
                                    [0, 0, 0, 0],
                                    [0, 0, 0, 0]  ]
    )
individual_fit_enough(population)
print(population)
select(population, calculate_fitness)
print('goal', genetic_search(population, calculate_fitness))
```

```
--->  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
--->  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
--->  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
--->  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[[1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [1, 1, 1, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0], [1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [1, 1,
1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
goal [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

[123]:

[124]:

[125]:

[138]:

[ ]:

[ ]:

[ ]: