Name: **Kunal Sinha** Batch: **Swarm R1- G2** Roll no: **2K17/CO/164**

# Experiment 7

**Aim:** Write a program to implement the Genetic algorithm.

## Theory:

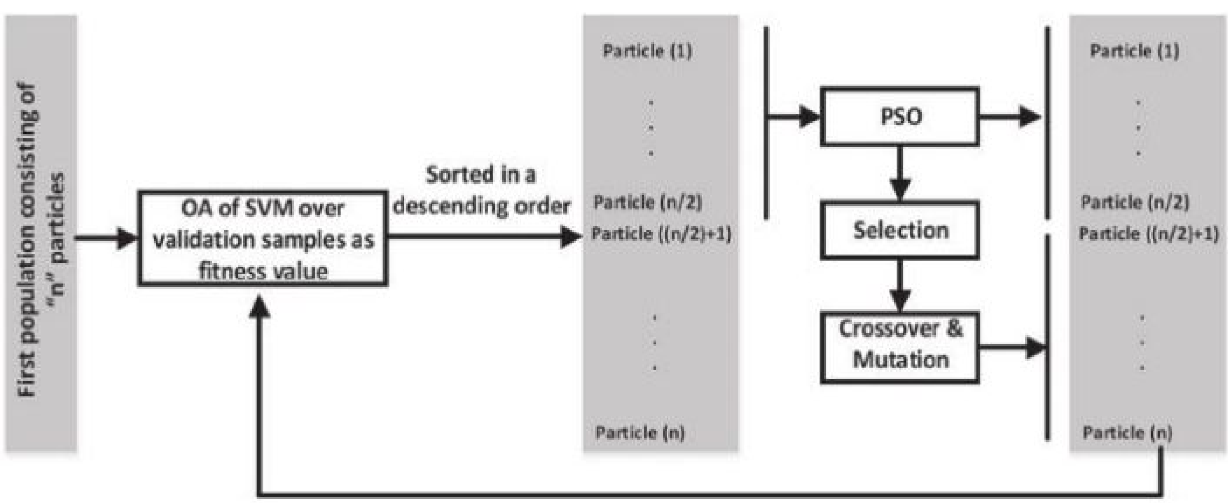
A genetic algorithm is a search heuristic that is inspired by Charles Darwin's theory of natural evolution. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation.

The process of natural selection starts with the selection of fittest individuals from a population. They produce offspring which inherit the characteristics of the parents and will be added to the next generation. If parents have better fitness, their offspring will be better than parents and have a better chance at surviving. This process keeps on iterating and at the end, a generation with the fittest individuals will be found. This notion can be applied for a search problem. We consider a set of solutions for a problem and select the set of best ones out of them. Five phases are considered in a genetic algorithm.

1. Initial population 2. Fitness function 3. Selection 4.Crossover 5.Mutation

## Algorithm:

1. Generate the initial population
2. Compute fitness
3. REPEAT
4. Selection
5. Crossover
6. Mutation
7. Compute fitness
8. UNTIL population has converged



## Source Code:

**genetic.py**

```
import numpy

def cal_pop_fitness(equation_inputs, pop):
    fitness = numpy.sum(pop*equation_inputs, axis=1)
    return fitness

def select_mating_pool(pop, fitness, num_parents):
    parents = numpy.empty((num_parents, pop.shape[1]))
    for parent_num in range(num_parents):
        max_fitness_idx = numpy.where(fitness == numpy.max(fitness))
        max_fitness_idx = max_fitness_idx[0][0]
        parents[parent_num, :] = pop[max_fitness_idx, :]
        fitness[max_fitness_idx] = -99999999999
    return parents

def crossover(parents, offspring_size):
    offspring = numpy.empty(offspring_size)
    crossover_point = numpy.uint8(offspring_size[1]/2)

    for k in range(offspring_size[0]):
        parent1_idx = k%parents.shape[0]
        parent2_idx = (k+1)%parents.shape[0]
        offspring[k, 0:crossover_point] = parents[parent1_idx, 0:crossover_point]
        offspring[k, crossover_point:] = parents[parent2_idx, crossover_point:]
    return offspring

def mutation(offspring_crossover, num_mutations=1):
    mutations_counter = numpy.uint8(offspring_crossover.shape[1] / num_mutations)
    for idx in range(offspring_crossover.shape[0]):
        gene_idx = mutations_counter - 1
        for mutation_num in range(num_mutations):
            random_value = numpy.random.uniform(-1.0, 1.0, 1)
            offspring_crossover[idx, gene_idx] = offspring_crossover[idx, gene_idx] +
random_value
            gene_idx = gene_idx + mutations_counter
    return offspring_crossover
```

**Main.py**

```
import numpy
import genetic
equation_inputs = [4,-2,3.5,5,-11,-4.7]
num_weights = len(equation_inputs)
sol_per_pop = 8
num_parents_mating = 4
pop_size = (sol_per_pop,num_weights)
new_population = numpy.random.uniform(low=-4.0, high=4.0, size=pop_size)
print(new_population)
best_outputs = []
num_generations = 10
for generation in range(num_generations):
    print("Generation : ", generation)
    fitness = genetic.cal_pop_fitness(equation_inputs, new_population)
    print("Fitness")
    print(fitness)
    best_outputs.append(numpy.max(numpy.sum(new_population*equation_inputs, axis=1)))
    print("Best result : ", numpy.max(numpy.sum(new_population*equation_inputs, axis=1)))
    parents = genetic.select_mating_pool(new_population, fitness,num_parents_mating)
    print("Parents")
    print(parents)
    offspring_crossover = genetic.crossover(parents,
                                    offspring_size=(pop_size[0]-parents.shape[0], num_weights))
    print("Crossover")
    print(offspring_crossover)
    offspring_mutation =genetic.mutation(offspring_crossover, num_mutations=2)
    print("Mutation")
    print(offspring_mutation)
    new_population[0:parents.shape[0], :] = parents
    new_population[parents.shape[0]:, :] = offspring_mutation

fitness = genetic.cal_pop_fitness(equation_inputs, new_population)
best_match_idx = numpy.where(fitness == numpy.max(fitness))
print("Best solution : ", new_population[best_match_idx, :])
print("Best solution fitness : ", fitness[best_match_idx])

import matplotlib.pyplot
matplotlib.pyplot.plot(best_outputs)
matplotlib.pyplot.xlabel("Iteration")
matplotlib.pyplot.ylabel("Fitness")
matplotlib.pyplot.show()
```

# Output:

**For 10 generations**

**Initial population**

```
C:\Users\Admin\Desktop>python geneticmain.py
[[ 0.10437277 -3.06269698  2.61504437 -1.56204928 -0.77732296 -2.80023349]
 [ 1.90703394 -0.18198742  3.98895785  2.86320899  0.88686627 -2.59161741]
 [ 3.31783698  1.40325753 -3.00092359 -1.24947539  0.88036932 -3.74177034]
 [-3.57553258  3.75161393 -1.43787474  3.53667965  1.54109297  3.79998988]
 [-2.96670332  1.31053164 -3.63234149  0.40062619 -2.71984275 -2.56084743]
 [ 0.06446086 -0.73115168 -1.15465806 -1.60236106  1.27753059 -1.32643179]
 [ 1.07586711  3.28407854 -0.76774762 -0.96239531  0.71842609 -1.6781868 ]
 [ 3.02939884 -3.10276914 -1.3491814  -3.36410929  3.50079599  1.62793956]]
```

**1st Generation**

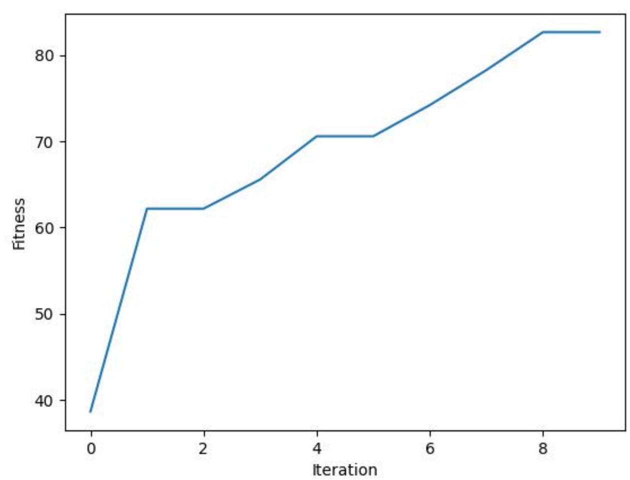
```
Generation :  0
Fitness
[ 29.59694383  38.69458088   1.61648135 -43.96649653  16.75631228
 -18.15156873  -9.77899084 -49.37961943]
Best result :  38.6945808806662
Parents
[[ 1.90703394 -0.18198742  3.98895785  2.86320899  0.88686627 -2.59161741]
 [ 0.10437277 -3.06269698  2.61504437 -1.56204928 -0.77732296 -2.80023349]
 [-2.96670332  1.31053164 -3.63234149  0.40062619 -2.71984275 -2.56084743]
 [ 3.31783698  1.40325753 -3.00092359 -1.24947539  0.88036932 -3.74177034]]
Crossover
[[ 1.90703394 -0.18198742  3.98895785 -1.56204928 -0.77732296 -2.80023349]
 [ 0.10437277 -3.06269698  2.61504437  0.40062619 -2.71984275 -2.56084743]
 [-2.96670332  1.31053164 -3.63234149 -1.24947539  0.88036932 -3.74177034]
 [ 3.31783698  1.40325753 -3.00092359  2.86320899  0.88686627 -2.59161741]]
Mutation
[[ 1.90703394 -0.18198742  3.33085439 -1.56204928 -0.77732296 -3.4889268 ]
 [ 0.10437277 -3.06269698  3.04059286  0.40062619 -2.71984275 -2.7838638 ]
 [-2.96670332  1.31053164 -4.50093676 -1.24947539  0.88036932 -3.99398814]
 [ 3.31783698  1.40325753 -2.22932092  2.86320899  0.88686627 -1.92043003]]
```

**10th Generation**

```
Generation :  9
Fitness
[82.6340379  81.27575319 78.24350761 75.28014413 77.55408117 76.10816489
 69.72476571 80.80394507]
Best result :  82.6340378950662
Parents
[[ 1.90703394 -0.18198742  4.01069215  0.40062619 -2.71984275 -6.102788  ]
 [ 1.90703394 -0.18198742  4.06366855  0.40062619 -2.71984275 -5.77434074]
 [ 1.90703394 -0.18198742  3.96449148  0.40062619 -2.71984275 -5.74781131]
 [ 1.90703394 -0.18198742  3.67155091  0.40062619 -2.71984275 -5.42118461]]
Crossover
[[ 1.90703394 -0.18198742  4.01069215  0.40062619 -2.71984275 -5.77434074]
 [ 1.90703394 -0.18198742  4.06366855  0.40062619 -2.71984275 -5.74781131]
 [ 1.90703394 -0.18198742  3.96449148  0.40062619 -2.71984275 -5.42118461]
 [ 1.90703394 -0.18198742  3.67155091  0.40062619 -2.71984275 -6.102788  ]]
Mutation
[[ 1.90703394 -0.18198742  4.27942301  0.40062619 -2.71984275 -5.870571  ]
 [ 1.90703394 -0.18198742  3.16050915  0.40062619 -2.71984275 -5.07097764]
 [ 1.90703394 -0.18198742  4.13891506  0.40062619 -2.71984275 -5.39482972]
 [ 1.90703394 -0.18198742  3.64479644  0.40062619 -2.71984275 -6.58908534]]
```

**Best solution**

```
Best solution :  [[[ 1.90703394 -0.18198742  3.64479644  0.40062619 -2.71984275
   -6.58908534]]]
Best solution fitness :  [83.63900039]
```




## Finding and Learnings:

We have successfully implemented the Genetic Algorithm (GA) in python. The fitness function determines how fit an individual is (the ability of an individual to compete with other individuals). It gives a fitness score to each individual. The probability that an individual will be selected for reproduction is based on its fitness score.