|  |  |
| --- | --- |
| Algorithm: Genetic Algorithm | |
| USN : 1MS17CS144 | NAME : Zaifa Khan |
| USN : 1MS17CS146 | NAME : Priya S. |

**Description of the Algorithm:**

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: Initial population, Fitness function, Selection, Crossover, Mutation.

**Algorithm Pseudocode:**

Start

1. Generate initial population.
2. Compute fitness for initial population.
3. Perform **selection** of fittest individuals.
4. Create offspring by performing **mating** of parents.
5. Perform mutation from the offspring with low probability, ensuring diversity.
6. Compute fitness.
7. Repeat from step 3 until population satisfies the termination condition.

**Data set Used: (Attach Screen shot of the few rows)**

No specific dataset was used for the Genetic Algorithm. An initial set of values randomly chosen were picked up, as genes, to create an initial population, whose fitness will then be calculated.

**Challenges faced during the implementation of the program:**

Genetic Algorithm is not available as a standard python library implementation. The entire algorithm must be written from initial stage.

**Code:**

import numpy as np

from numpy.random import randint

from random import random as rnd

from random import gauss, randrange

#Initial Population Individual function

def individual(number\_of\_genes, upper\_limit, lower\_limit):

individual=[round(rnd()\*(upper\_limit-lower\_limit)

+lower\_limit,1) for x in range(number\_of\_genes)]

return individual

#Population function

def population(number\_of\_individuals,

number\_of\_genes, upper\_limit, lower\_limit):

return [individual(number\_of\_genes, upper\_limit, lower\_limit)

for x in range(number\_of\_individuals)]

#Fitness Calculation

def fitness\_calculation(individual):

fitness\_value = sum(individual)

return fitness\_value

#Roulette Wheel Selection

def roulette(cum\_sum, chance):

veriable = list(cum\_sum.copy())

veriable.append(chance)

veriable = sorted(veriable)

return veriable.index(chance)

def selection(generation, method='Roulette Wheel'):

generation['Normalized Fitness'] = \

sorted([generation['Fitness'][x]/sum(generation['Fitness'])

for x in range(len(generation['Fitness']))], reverse = True)

generation['Cumulative Sum'] = np.array(

generation['Normalized Fitness']).cumsum()

if method == 'Roulette Wheel':

selected = []

for x in range(len(generation['Individuals'])//2):

selected.append(roulette(generation

['Cumulative Sum'], rnd()))

while len(set(selected)) != len(selected):

selected[x] = \

(roulette(generation['Cumulative Sum'], rnd()))

selected = {'Individuals':

[generation['Individuals'][int(selected[x])]

for x in range(len(generation['Individuals'])//2)]

,'Fitness': [generation['Fitness'][int(selected[x])]

for x in range(

len(generation['Individuals'])//2)]}

return selected

#Pairing Function

def pairing(elit, selected, method = 'Weighted Random'):

individuals = [elit['Individuals']]+selected['Individuals']

fitness = [elit['Fitness']]+selected['Fitness']

if method == 'Weighted Random':

normalized\_fitness = sorted(

[fitness[x] /sum(fitness)

for x in range(len(individuals)//2)], reverse = True)

cummulitive\_sum = np.array(normalized\_fitness).cumsum()

parents = []

for x in range(len(individuals)//2):

parents.append(

[individuals[roulette(cummulitive\_sum,rnd())],

individuals[roulette(cummulitive\_sum,rnd())]])

while parents[x][0] == parents[x][1]:

parents[x][1] = individuals[

roulette(cummulitive\_sum,rnd())]

return parents

#Mating Function

def mating(parents, method='Single Point'):

if method == 'Single Point':

pivot\_point = randint(1, len(parents[0]))

offsprings = [parents[0] \

[0:pivot\_point]+parents[1][pivot\_point:]]

offsprings.append(parents[1]

[0:pivot\_point]+parents[0][pivot\_point:])

return offsprings

#Mutations

def mutation(individual, upper\_limit, lower\_limit, muatation\_rate=2,

method='Reset', standard\_deviation = 0.001):

gene = [randint(0, 7)]

for x in range(muatation\_rate-1):

gene.append(randint(0, 7))

while len(set(gene)) < len(gene):

gene[x] = randint(0, 7)

mutated\_individual = individual.copy()

if method == 'Gauss':

for x in range(muatation\_rate):

mutated\_individual[x] = \

round(individual[x]+gauss(0, standard\_deviation), 1)

if method == 'Reset':

for x in range(muatation\_rate):

mutated\_individual[x] = round(rnd()\* \

(upper\_limit-lower\_limit)+lower\_limit,1)

return mutated\_individual

#Creating next generation

def next\_generation(gen, upper\_limit, lower\_limit):

elit = {}

next\_gen = {}

elit['Individuals'] = gen['Individuals'].pop(-1)

elit['Fitness'] = gen['Fitness'].pop(-1)

selected = selection(gen)

parents = pairing(elit, selected)

offsprings = [[[mating(parents[x])

for x in range(len(parents))]

[y][z] for z in range(2)]

for y in range(len(parents))]

offsprings1 = [offsprings[x][0]

for x in range(len(parents))]

offsprings2 = [offsprings[x][1]

for x in range(len(parents))]

unmutated = selected['Individuals']+offsprings1+offsprings2

mutated = [mutation(unmutated[x], upper\_limit, lower\_limit)

for x in range(len(gen['Individuals']))]

unsorted\_individuals = mutated + [elit['Individuals']]

unsorted\_next\_gen = \

[fitness\_calculation(mutated[x])

for x in range(len(mutated))]

unsorted\_fitness = [unsorted\_next\_gen[x]

for x in range(len(gen['Fitness']))] + [elit['Fitness']]

sorted\_next\_gen = \

sorted([[unsorted\_individuals[x], unsorted\_fitness[x]]

for x in range(len(unsorted\_individuals))],

key=lambda x: x[1])

next\_gen['Individuals'] = [sorted\_next\_gen[x][0]

for x in range(len(sorted\_next\_gen))]

next\_gen['Fitness'] = [sorted\_next\_gen[x][1]

for x in range(len(sorted\_next\_gen))]

gen['Individuals'].append(elit['Individuals'])

gen['Fitness'].append(elit['Fitness'])

return next\_gen

#Termination

def fitness\_similarity\_chech(max\_fitness, number\_of\_similarity):

result = False

similarity = 0

for n in range(len(max\_fitness)-1):

if max\_fitness[n] == max\_fitness[n+1]:

similarity += 1

else:

similarity = 0

if similarity == number\_of\_similarity-1:

result = True

return result

# Generations and fitness values will be written to this file

Result\_file = '/content/drive/My Drive/GA\_Results.txt'

# Creating the First Generation

def first\_generation(pop):

fitness = [fitness\_calculation(pop[x])

for x in range(len(pop))]

sorted\_fitness = sorted([[pop[x], fitness[x]]

for x in range(len(pop))], key=lambda x: x[1])

population = [sorted\_fitness[x][0]

for x in range(len(sorted\_fitness))]

fitness = [sorted\_fitness[x][1]

for x in range(len(sorted\_fitness))]

return {'Individuals': population, 'Fitness': sorted(fitness)}

pop = population(20,8,1,0)

gen = []

gen.append(first\_generation(pop))

fitness\_avg = np.array([sum(gen[0]['Fitness'])/

len(gen[0]['Fitness'])])

fitness\_max = np.array([max(gen[0]['Fitness'])])

res = open(Result\_file, 'a')

res.write('\n'+str(gen)+'\n')

res.close()

finish = False

while finish == False:

if len(gen)==10:

finish=true

break

gen.append(next\_generation(gen[-1],1,0))

fitness\_avg = np.append(fitness\_avg, sum(

gen[-1]['Fitness'])/len(gen[-1]['Fitness']))

fitness\_max = np.append(fitness\_max, max(gen[-1]['Fitness']))

res = open(Result\_file, 'a')

res.write('\n'+str(gen[-1])+'\n')

res.close()

**Output: (Screen shots)**

**A close up of a piece of paper

Description automatically generatedA screenshot of text

Description automatically generated**

GA\_Results.txt

**References:**

1. <https://towardsdatascience.com/continuous-genetic-algorithm-from-scratch-with-python-ff29deedd099>
2. <https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3>