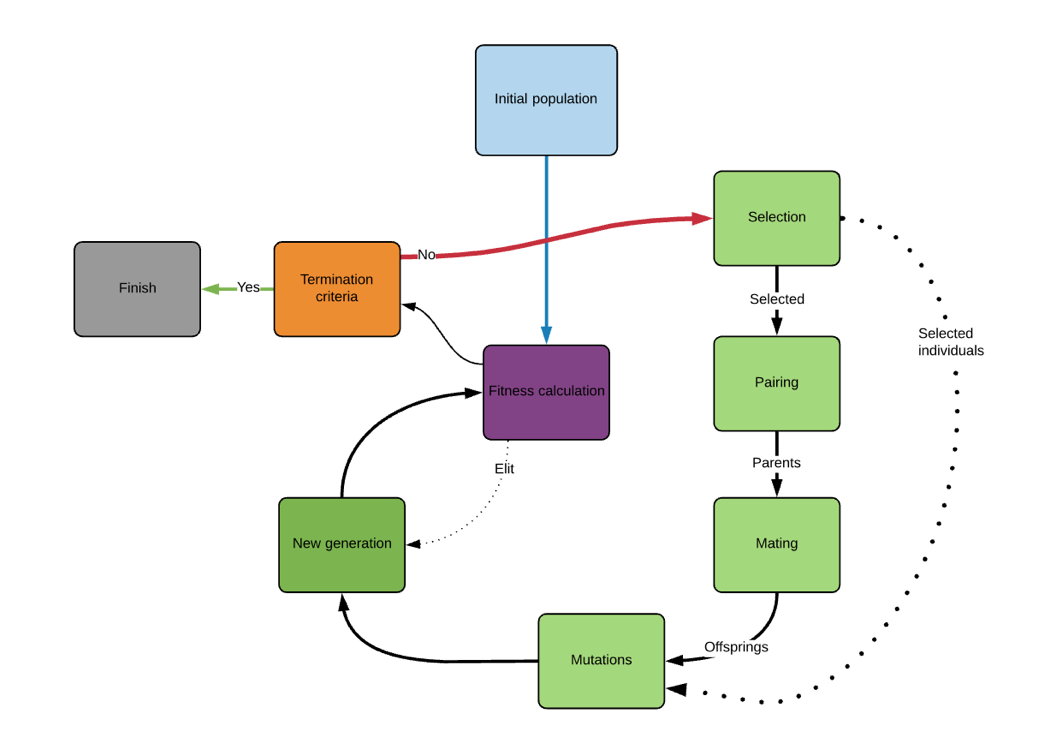
|  |  |
| --- | --- |
| Algorithm: Genetic Algorithm | |
| USN : 1MS17CS143 | NAME : Sathvik K P |
| USN : 1MS17CS148 | NAME : Sathvik B |

**Description of the Algorithm:**

Genetic algorithm is a powerful optimization technique that was inspired by nature. Genetic algorithms mimic evolution to find the best solution. Unlike most optimization algorithms, genetic algorithms do not use derivatives to find the minima. One of the most significant advantages of genetic algorithms is their ability to find a global minimum without getting stuck in local minima. Randomness plays a substantial role in the structure of genetic algorithms, and it is the main reason genetic algorithms keep searching the search space.

Genetic algorithms create an initial population of randomly generated candidate solutions, these candidate solutions are evaluated, and their fitness value is calculated. The fitness value of a solution is the numeric value that determines how good a solution is, higher the fitness value better the solution.

If the initial population does not meet the requirements of the termination criteria, genetic algorithm creates the next generation. The first genetic operation is Selection; in this operation, the individuals that are going to be moving on to the next generation are selected. After the selection process, Pairing operation commences. Pairing operation pairs the selected individuals two by two for the Mating operation. The Mating operation takes the paired parent individuals and creates offsprings, which will be replacing the individuals that were not selected in the Selection operation, so the next generation has the same number of individuals as the previous generation. This process is repeated until the termination criteria is met.



**Algorithm Pseudocode:**

START  
Generate the initial population  
Compute fitness  
REPEAT  
 Selection  
 Crossover  
 Mutation  
 Compute fitness  
UNTIL population has converged  
STOP

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

1. Lack of familiarity with G.A
2. Results difficult to analyze
3. Randomness associated with mutations sometimes doesn’t give optimal solutions if less number of generations are used.
4. Most cases have only one minima hence, other methods are useful.

**Code:**

import numpy as np

from numpy.random import randint

from random import random as rnd

from random import gauss, randrange

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

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)]

def fitness\_calculation(individual):

    fitness\_value = sum(individual)

    return fitness\_value

def selection(generation, method='Fittest Half'):

    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 == 'Fittest Half':

        selected\_individuals = [generation['Individuals'][-x-1]

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

        selected\_fitnesses = [generation['Fitness'][-x-1]

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

        selected = {'Individuals': selected\_individuals,

                    'Fitness': selected\_fitnesses}

    return selected

def pairing(elit, selected, method = 'Fittest'):

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

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

    if method == 'Fittest':

        parents = [[individuals[x],individuals[x+1]]

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

    return parents

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

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

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

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 = '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 = 0

while finish <=20:

    finish+=1

    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')

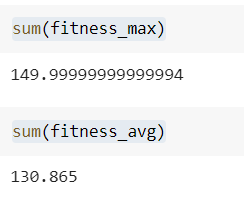
    res.write('\n'+str(gen[-1])+'\n')

    res.close()

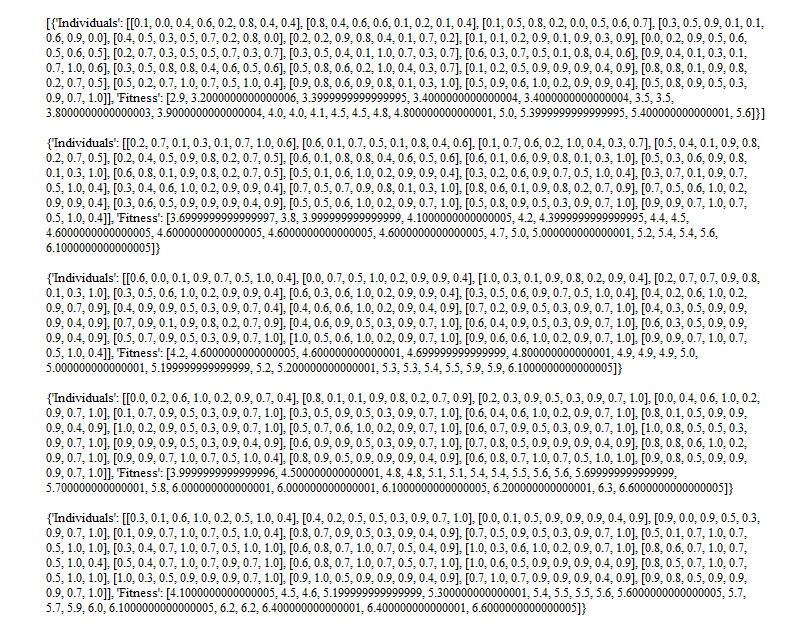
sum(fitness\_max)

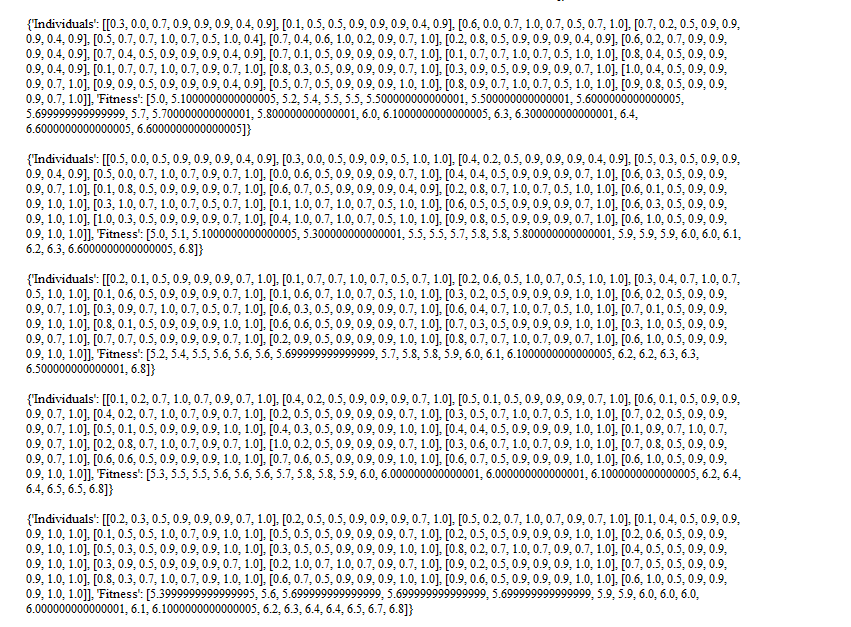
sum(fitness\_avg)

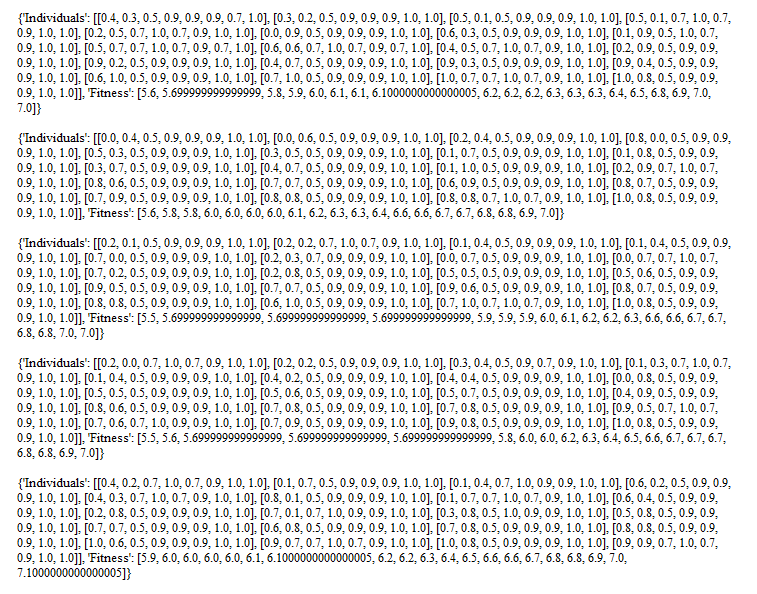
**Output: (Screen shots)**

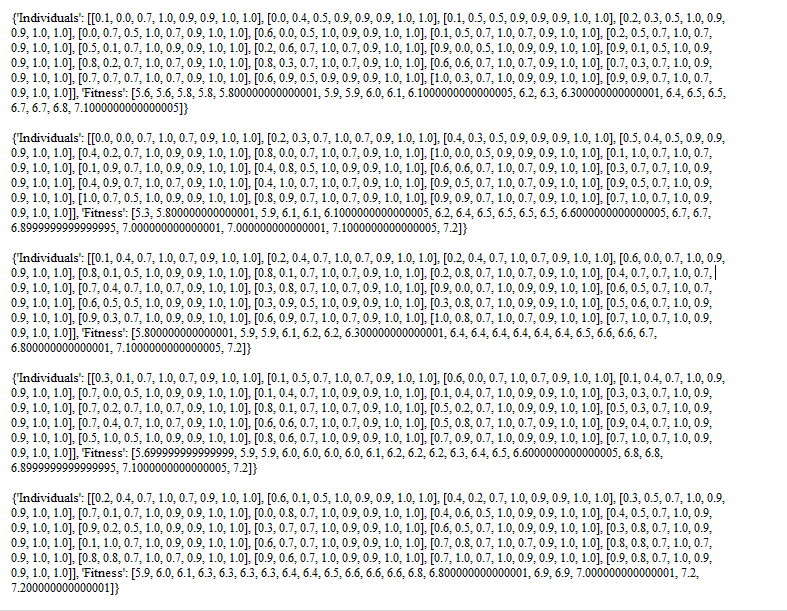


G.A results.txt









**References:**

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