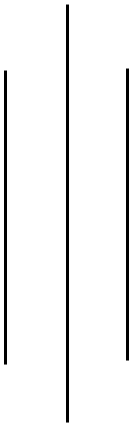
TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS

A
LAB REPORT
ON

GENETIC ALGORITHM



SUBMITTED BY:

Diwas Adhikari **(076BEI014)**

SUBMITTED TO:

Department of Electronics & Computer Engineering
Pulchowk Campus, IOE

LAB NUMBER

6

SUBMISSION DATE

August 7, 2023

(For course completion of BEI – Artificial Intelligence)

OBJECTIVES

- To understand how genetic algorithm works and get used to its terminologies
- To solve problems using genetic algorithm and evaluate its performance

THEORY

A genetic algorithm is a metaheuristic inspired by the process of natural selection by Charles Darwin that belongs to the larger class of other evolutionary algorithm. Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on biologically inspired operators such as selection, mutation and crossover.

Phases of Genetic Algorithm

The selection of the fittest members of a population is the first step in the process of natural selection. They give birth to children who carry on their parents' traits and join the following generation. Parents who are more physically fit will produce children who will outperform them and have a higher chance of surviving. The fittest generation will eventually emerge as a result of this iterative procedure.This idea can be used to solve a search difficulty. We take into account a number of potential solutions to a problem and choose the finest ones.

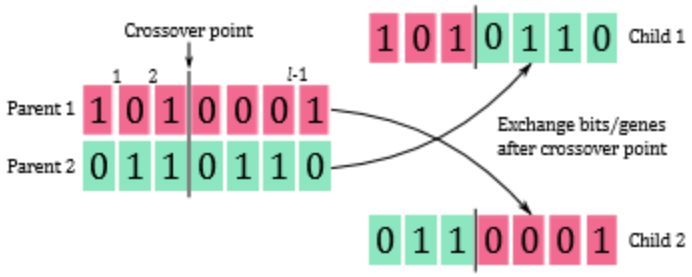
The five major phases of genetic algorithm are:

Initial Population: Each member of the population, which serves as the starting point for the genetic algorithm, is a potential solution to the problem. A chromosome, which is a full solution, is made up of a set of parameters called genes that characterize each person. In a genetic algorithm, a string, typically in binary format, is used to represent a person's genes. Encoding is the method used to represent genes in a chromosome.

Fitness: The fitness function is in charge of judging the effectiveness or suitability of each person's answer to the issue at hand. It determines an individual's fitness score depending on how well its answer satisfies the required requirements. An individual is more likely to be chosen for reproduction the higher their fitness score. Who will succeed in the population and hence have the best chance of achieving the desired result is ultimately determined by the fitness function.

Selection: The goal of a genetic algorithm's selection phase is to locate and pick the people who are most suited for the job at hand. This entails selecting people who are the fittest based on how effectively they perform within the specified problem domain. Depending on their relative fitness scores, two pairs of people, or parents, will be picked as part of the selection process. Higher fitness individuals are typically more likely to be chosen for reproduction because they are thought to be better able to pass on their favorable features to the next generation.

Crossover: To produce kids with a novel gene combination, two parents exchange genetic information through the process of crossover. The crossover is a crucial operator in genetic algorithms that encourages variation and gives fresh ideas to the population. By introducing novel variations, crossover aims to create kids that combine the best traits of both parents and enhance the fitness of the offspring.



Mutation: To preserve genetic variety from one generation of a population to the next, mutation is a genetic operator used in genetic algorithms. It causes arbitrary modifications in a person's chromosome, such as changing a bit from 0 to 1 or vice versa, and may possibly bring new solutions to the issue that were absent from the starting population.

The algorithm terminates if the population has converged which suggests that it does not produce offspring which are significantly different from the previous generation.

Problem Scenario - 'Painter Robot'

Let's imagine, we have a painter robot similar to the robot which picked up cans in the lectures. We will use this robot to paint the floor of a room. To make it interesting, the painter starts at a random place in the room, and paints continuously. We will also imagine that there is exactly enough paint to cover the floor. This means that it is wasteful to visit the same spot more than once or to stay in the same place. To see if there is a optimal set of rules for the painter to follow, you will create a genetic algorithm.

As inputs, this function receives:-

1. A chromosome: A 1x54 array of numbers between 0 and 3 that shows how to respond (0: no turn, 1:turn left, 2:turn right, 3: random turn left/right) in each of the 54 possible states. The state is the state of the squares forward/left/right and the current square. Let [c, f, l, r] denote states of the current square, forward square, left square and right square respectively. Write 0 for empty, 1 for wall/obstruction and 2 for painted.

Note that $c \in \{0, 2\}$ and $f, l, r \in \{0, 1, 2\}$ so there are $2 \times 33 = 54$ possible states.

2. An environment: A 2D array representing a rectangular room. Empty (paintable) space is represented by a zero, while furniture or extra walls are represented by ones. Outside walls are automatically created by painter_play().

The function painter_play() then uses the rule set to guide a painter, initially placed in the room with a random position and direction, until the paint can is empty. Note that the painter does not move when it tries to walk into a wall or furniture. The efficiency (total fraction of paintable space covered) is then given as an output, as well as the X-Y trajectory (i.e. the positions of the painter at each time step) of the painter. To see that the painter works, you can try passing it an empty room for an environment and a trivial chromosome. For example, a chromosome consisting of all 3's produces a kind of random walk.

PYTHON CODE (Painter Robot)

```
# Painter Robot Using Genetic Algorithm - Diwas Adhikari (076BEI014)
import random ;
import numpy as np ;
import matplotlib.pyplot as plt ;
# - The chromosome is of 50*54 dimensions with 50 random chromosomes to evolve over 200 generations.
totalGenerations = 200 ;
populationSize = 50 ;
lengthChromosome = 54 ;
playsPerChromosome = 5 ;
# - The dimensions of the empty room to be painted is 20*40 units.
roomWidth = 20 ;
roomHeight = 40 ;
room = [[0 for _ in range(roomWidth)] for _ in range(roomHeight)] ;
# - Function to display the room
def showRoom(room):
    cmap = {0: "green", 1: "yellow", 2: "brown"} ;
    plt.figure(figsize=(2,4)) ;
    for y in range(len(room)):
        for x in range(len(room[0])):
            value = room[y][x] ;
            color = cmap[value] ;
            plt.fill_between([x, x + 1], y, y + 1, color=color) ;
    plt.xlim(0, len(room[0])) ;
    plt.ylim(0, len(room)) ;
    plt.gca().invert_yaxis() ;
    plt.axis("off") ;
    plt.show() ;
showRoom(room) ;

# - Initialize a random population with random chromosomes.
population = [] ;
for i in range(populationSize):
    newChromosome = [np.random.randint(0, 3) for _ in
range(lengthChromosome)] ;
    population.append(newChromosome) ;

# - Calculate fitness based on the number of tiles it can paint for the room
def calculateFitness(chromosome, room):
    score = 0 ;
    position = [0, 0] ;
    painted = set() ;
    for movement in chromosome:
        if movement == 0: newPosition = [position[0], position[1]+1] ;
        elif movement == 1: newPosition = [position[0]-1, position[1]] ;
```

```
        elif movement == 2: newPosition = [position[0]+1, position[1]] ;
        else: newPosition = [position[0] + np.random.choice([-1, 1]), position[1]] ;
        # Check if the robot is within bounds
        if (newPosition[0] <= 0 or newPosition[0] >= roomHeight
            or newPosition[1] < 0 or newPosition[1] >= roomWidth): continue ;
        # Check if that tile is already painted
        if tuple(newPosition) in painted: continue ;
        # Update the scores if it is actually a new tile that is never painted before
        painted.add(tuple(newPosition)) ;
        score += room[newPosition[0]][newPosition[1]] == 0 ;
        position = newPosition ;
    return score ;

# - Use genetic operators to evolve the population.
def evolvePopulation(population, room, mutationProbability):
    # Evaluate fitness of each chromosome in the population
    fitness = [] ;
    for chromosome in population:
        totalEfficiency = 0 ;
        for i in range(playsPerChromosome):
            totalEfficiency += calculateFitness(chromosome, room) ;
        avgEfficiency = totalEfficiency/playsPerChromosome ;
        fitness.append(avgEfficiency) ;
    # Selection
    parents = [] ;
    for i in range(populationSize // 2):
        selectionQuota = random.sample(fitness, 5) ;
        selected = fitness.index(max(selectionQuota)) ;
        parents.append(population[selected]) ;
    # Crossover
    newGeneration = [] ;
    for i in range(populationSize):
        parent1 = random.choice(parents) ;
        parent2 = random.choice(parents) ;
        offsprings = [] ;
        # Use a crossover point to break and swap bits of chromosome
        crossoverPoint = np.random.randint(0, lengthChromosome) ;
        for j in range(lengthChromosome):
            if j <= crossoverPoint: offsprings.append(parent1[j]) ;
            else: offsprings.append(parent2[j]) ;
            # Mutation
            if np.random.random() < mutationProbability:
                    offsprings[j] = random.randint(0, 3) ;
        newGeneration.append(offsprings) ;
    return newGeneration ;
```

```
# Perform evolution over multiple generations
def geneticTraining(population, room, mutationProbability, totalGenerations):
    for generation in range(totalGenerations):
        population = evolvePopulation(population, room, mutationProbability) ;
        fitness = [] ;
        for chromosome in population:
            x = calculateFitness(chromosome, room) ;
            fitness.append(x) ;
        averageFitness = sum(fitness)/len(fitness) ;
        print("Generation", generation, "-> Average fitness :", averageFitness) ;
    return population ;

# Deploy robot in open field
population = geneticTraining(population, room, 0.025, totalGenerations) ;
maxFitness = -99999 ;
chosenOne = 0 ;
for i, chromosome in enumerate(population):
    x = calculateFitness(chromosome, room) ;
    if x > maxFitness: chosenOne = i ;
fittestChromosome = population[chosenOne] ;
print("The fittest chromosome is ", fittestChromosome) ;

# - Use the best chromosome to track the path of the robot for plotting it later
def trackPathInRoom(room, fittestChromosome):
    # '0' is unpainted - '1' is wall - '2' is painted
    robotPosition = [0, 0] ;
    room[0][0] = 2 ;
    for movement in fittestChromosome:
        if movement == 0: # Forward
            if robotPosition[1] >= roomHeight:
                continue ;
            robotPosition[1] += 1 ;
        elif movement == 1: # Left
            if robotPosition[0] <= 0:
                continue ;
            robotPosition[0] -= 1 ;
        elif movement == 2: # Right
            if robotPosition[0] >= roomWidth:
                continue ;
            robotPosition[0] += 1 ;
        else: # Random left or right
            if robotPosition[0] >= roomWidth or robotPosition[0] <= 0:
                continue ;
            robotPosition[0] += np.random.choice([-1, 1]) ;
        room[robotPosition[0]][robotPosition[1]] = 2 ;
    return room ;
```

```
room = trackPathInRoom(room, fittestChromosome) ;
showRoom(room) ;

# - Put some obstacles to check the performance of the fittest genome.
obstacleRoom = [[0 for _ in range(roomWidth)] for _ in range(roomHeight)] ;
for i in range(100):
    posX = np.random.randint(1, roomWidth-1) ;
    posY = np.random.randint(1, roomHeight-1) ;
    obstacleRoom[posY][posX] = 1 ;
showRoom(obstacleRoom) ;
population = geneticTraining(population, obstacleRoom, 0.025,totalGenerations) ;
maxFitness = -99999 ;
chosenOne = 0 ;
for i, chromosome in enumerate(population):
    x = calculateFitness(chromosome, obstacleRoom) ;
    if x > maxFitness:
        chosenOne = i ;
fittestChromosome = population[chosenOne] ;
print("The fittest chromosome is ", fittestChromosome) ;
obstacleRoom = trackPathInRoom(obstacleRoom, fittestChromosome) ;
showRoom(obstacleRoom) ;
```

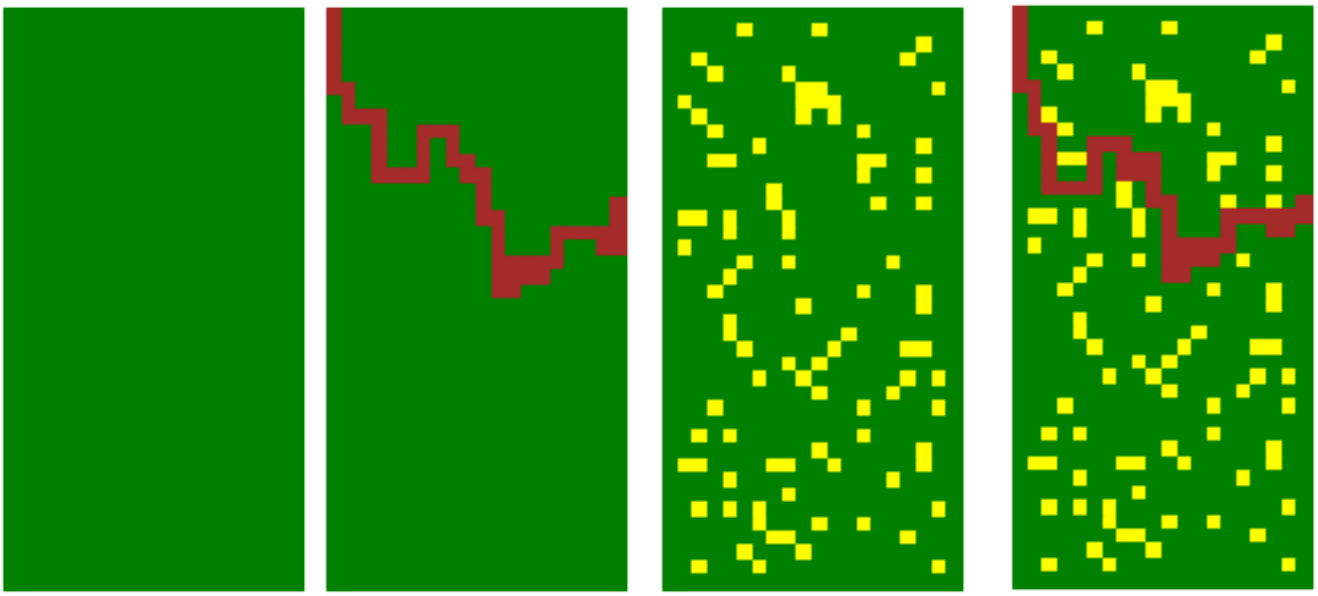
OBSERVATIONS

1) Painter Robot – Average Fitness (after training for 200 generations)

```
1 population = geneticTraining(population, room, 0.025, totalGenerations) ;
Generation 181 -> Average fitness : 53.04
Generation 182 -> Average fitness : 53.1
Generation 183 -> Average fitness : 52.82
Generation 184 -> Average fitness : 53.1
Generation 185 -> Average fitness : 53.14
Generation 186 -> Average fitness : 53.06
Generation 187 -> Average fitness : 52.66
Generation 188 -> Average fitness : 52.7
Generation 189 -> Average fitness : 52.28
Generation 190 -> Average fitness : 52.8
Generation 191 -> Average fitness : 52.58
Generation 192 -> Average fitness : 52.68
Generation 193 -> Average fitness : 52.6
Generation 194 -> Average fitness : 53.14
Generation 195 -> Average fitness : 52.72
Generation 196 -> Average fitness : 52.74
Generation 197 -> Average fitness : 52.98
Generation 198 -> Average fitness : 52.82
Generation 199 -> Average fitness : 52.66
```

```
1 population = geneticTraining(population, obstacleRoom, 0.025, totalGenerations) ;
Generation 181 -> Average fitness : 51.66
Generation 182 -> Average fitness : 51.78
Generation 183 -> Average fitness : 51.86
Generation 184 -> Average fitness : 51.8
Generation 185 -> Average fitness : 51.84
Generation 186 -> Average fitness : 51.92
Generation 187 -> Average fitness : 51.5
Generation 188 -> Average fitness : 51.82
Generation 189 -> Average fitness : 51.9
Generation 190 -> Average fitness : 51.82
Generation 191 -> Average fitness : 51.74
Generation 192 -> Average fitness : 51.7
Generation 193 -> Average fitness : 51.82
Generation 194 -> Average fitness : 51.66
Generation 195 -> Average fitness : 51.86
Generation 196 -> Average fitness : 51.84
Generation 197 -> Average fitness : 51.9
Generation 198 -> Average fitness : 51.9
Generation 199 -> Average fitness : 51.7
```

2) Painter Robot – Trajectory / Path Plot



(a) Open Field (b) Painted Path (c) Obstructed Field (d) Optimized Path

DISCUSSION & CONCLUSION

In this lab session, we understood the basics and key terminologies associated to genetic algorithms and implemented genetic training methods and operators to visualize the painter robot problem. Fig. (b) and Fig. (d) has slight differences in the path of the robot since the obstacles are introduced in the field later. Thus, this also indicates a small decrease in efficiency and fitness of the genome as it can choose between less number of tiles and paint fewer tiles as well as it couldn't follow the similar path it travelled when there were no blocks as obstructions. Hence, genetic algorithm can be used for optimizing solutions based on improving the fitness value associated to a particular problem and evolving it throughout the next generations.

SUBMITTED BY:

Diwas Adhikari
076BEI014 – (Roll Number: 14)
Bachelor in Electronics & Communication Engineering (BEI)
076bei014.diwas@pcampus.edu.np
lunaticoda123@gmail.com