Name: Dhruv Bheda **SapID:** 60004200102

Div: B/B1

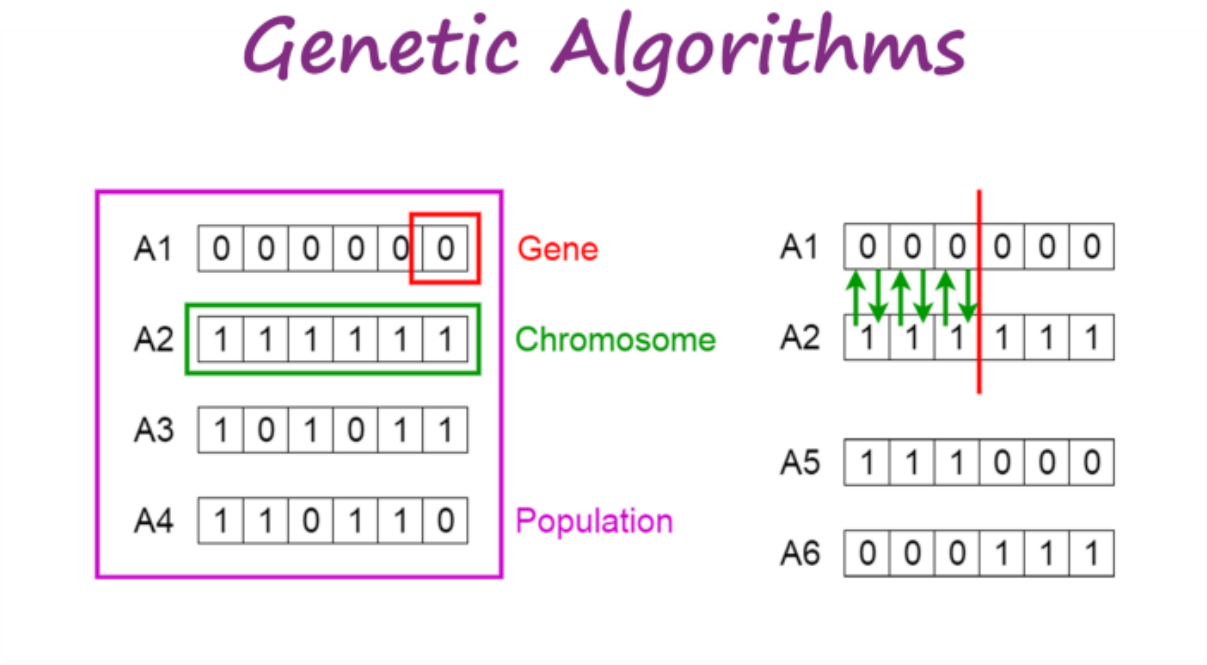
A.I.

Exp5

Aim: To study and implement 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.



Five phases are considered in a genetic algorithm.

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

Initial Population

The process begins with a set of individuals which is called a Population. Each individual is a solution to the problem you want to solve.An individual is characterized by a set of parameters (variables) known as Genes. Genes are joined into a string to form a Chromosome (solution).

Fitness Function

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.

Selection

The idea of selection phase is to select the fittest individuals and let them pass their genes to the next generation.Two pairs of individuals (parents) are selected based on their fitness scores. Individuals with high fitness have more chance to be selected for reproduction.

Crossover

Crossover is the most significant phase in a genetic algorithm. For each pair of parents to be mated, a crossover point is chosen at random from within the genes.

Mutation

In certain new offspring formed, some of their genes can be subjected to a mutation with a low random probability. This implies that some of the bits in the bit string can be flipped.

Code:

```
from numpy.random import randint

from numpy.random import rand

import math

def crossover(parent1, parent2, r_cross):

    child1, child2 = parent1.copy(), parent2.copy()

    r = rand()

    point = 0

    if r > r_cross:

        point = randint(1, len(parent1) - 2)

        child1 = parent1[:point] + parent2[point:]

        child2 = parent2[:point] + parent1[point:]

    return child1, child2, point
```

```
def mutate(chromosome, r_mut):
    for i in range(len(chromosome)):
        if rand() < r_mut:
            chromosome[i] = 1 - chromosome[i]
    return chromosome


def bin_to_dec(bin):
    decimal = 0
    for i in range(len(bin)):
        decimal += bin[i] * pow(2, 4 - i)
    return decimal


def dec_to_bin(dec):
    binaryVal = []
    while dec > 0:
        binaryVal.append(dec % 2)
        dec = math.floor(dec / 2)
    for _ in range(5 - len(binaryVal)):
        binaryVal.append(0)
    binaryVal = binaryVal[::-1]
```

```
    return binaryVal


def fitness_function(x):
    return pow(x, 2)
def genetic_algorithm(iterations, population_size, r_cross, r_mut):
    input = [randint(0, 32) for _ in range(population_size)]
    pop = [dec_to_bin(i) for i in input]
    for generation in range(iterations):
        print(f"\nGeneration : {generation+1}", end="\n\n")
        decimal = [bin_to_dec(i) for i in pop]
        fitness_score = [fitness_function(i) for i in decimal]
        f_by_sum = [
            fitness_score[i] / sum(fitness_score) for i in range(population_size)
        ]
        exp_cnt = [
            fitness_score[i] / (sum(fitness_score) / population_size)
            for i in range(population_size)
        ]
        act_cnt = [round(exp_cnt[i]) for i in range(population_size)]
        print(
```

```
        "SELECTION\n\nInitial Population\tDecimal Value\tFitness
Score\tFi/Sum\tExpected count\tActual Count"
    )
    for i in range(population_size):
        print(
            pop[i],
            "\t",
            decimal[i],
            "\t\t",
            fitness_score[i],
            "\t\t",
            round(f_by_sum[i], 2),
            "\t\t",
            round(exp_cnt[i], 2),
            "\t\t",
            act_cnt[i],
        )
    print("Sum : ", sum(fitness_score))
    print("Average : ", sum(fitness_score) / population_size)
    print("Maximum : ", max(fitness_score), end="\n")
```

```
        max_count = max(act_cnt)
        min_count = min(act_cnt)
        max_count_index = 0
        for i in range(population_size):
            if max_count == act_cnt[i]:
                max_count_index = i
                break
        for i in range(population_size):
            if min_count == act_cnt[i]:
                pop[i] = pop[max_count_index]
        crossover_children = list()
        crossover_point = list()
        for i in range(0, population_size, 2):
            child1, child2, point_of_crossover = crossover(pop[i], pop[i + 1],
r_cross)
            crossover_children.append(child1)
            crossover_children.append(child2)
            crossover_point.append(point_of_crossover)
            crossover_point.append(point_of_crossover)
        print(
```

```
        "\nCROSS OVER\n\nPopulation\t\tMate\t Crossover Point\t Crossover Population"
    )
    for i in range(population_size):
        if (i + 1) % 2 == 1:
            mate = i + 2
        else:
            mate = i
        print(
            pop[i],
            "\t",
            mate,
            "\t",
            crossover_point[i],
            "\t\t\t",
            crossover_children[i],
        )
    mutation_children = list()
    for i in range(population_size):
        child = crossover_children[i]
```

```
        mutation_children.append(mutate(child, r_mut))
    new_population = list()
    new_fitness_score = list()
    for i in mutation_children:
        new_population.append(bin_to_dec(i))
    for i in new_population:
        new_fitness_score.append(fitness_function(i))
    print("\nMUTATION\n\nMutation population\t New Population\t Fitness Score")
    for i in range(population_size):
        print(
            mutation_children[i],
            "\t",
            new_population[i],
            "\t\t",
            new_fitness_score[i],
        )
    print("Sum : ", sum(new_fitness_score))
    print("Maximum : ", max(new_fitness_score))
    pop = mutation_children
```

genetic_algorithm(iterations=2, population_size=4, r_cross=0.5, r_mut=0.05)

Output:

```
Maximum :  729

CROSS OVER

Population              Mate     Crossover Point         Crossover Population
[1, 0, 1, 0, 0]          2       1                       [1, 1, 0, 1, 1]
[1, 1, 0, 1, 1]          1       1                       [1, 0, 1, 0, 0]
[1, 1, 0, 1, 1]          4       0                       [1, 1, 0, 1, 1]
[1, 1, 0, 1, 1]          3       0                       [1, 1, 0, 1, 1]

MUTATION

Mutation population      New Population  Fitness Score
[1, 1, 0, 1, 0]          26              676
[1, 0, 1, 0, 0]          20              400
[1, 1, 0, 1, 1]          27              729
[1, 1, 0, 1, 1]          27              729
Sum :  2534
Maximum :  729

Generation : 2

SELECTION

Initial Population      Decimal Value   Fitness Score   Fi/Sum  Expected count  Actual Count
[1, 1, 0, 1, 0]          26              676             0.27            1.07            1
[1, 0, 1, 0, 0]          20              400             0.16            0.63            1
[1, 1, 0, 1, 1]          27              729             0.29            1.15            1
[1, 1, 0, 1, 1]          27              729             0.29            1.15            1
Sum :  2534
Average :  633.5
Maximum :  729

CROSS OVER

Population              Mate     Crossover Point         Crossover Population
[1, 1, 0, 1, 0]          2       1                       [1, 1, 0, 1, 0]
[1, 1, 0, 1, 0]          1       1                       [1, 1, 0, 1, 0]
[1, 1, 0, 1, 0]          4       1                       [1, 1, 0, 1, 0]
[1, 1, 0, 1, 0]          3       1                       [1, 1, 0, 1, 0]

MUTATION

Mutation population      New Population  Fitness Score
[1, 0, 0, 0, 0]          16              256
[1, 1, 0, 1, 0]          26              676
[1, 1, 0, 1, 0]          26              676
[1, 1, 0, 1, 0]          26              676
Sum :  2284
Maximum :  676
```

Conclusion:

Thus we successfully studied and applied Genetic Algorithm