**Experiment No 05**

**Aim:** Implementation of Genetic Algorithm

## Theory:

Genetic algorithms are optimization algorithms inspired by the process of natural selection. They involve evolving a population of potential solutions to a problem over multiple generations. Here's a brief overview:

1. **Initialization**: Start with a population of potential solutions (chromosomes) randomly generated.
2. **Selection**: Evaluate the fitness of each solution and select individuals to serve as parents based on their fitness. Solutions with higher fitness have a higher chance of being selected.
3. **Crossover (Recombination)**: Pair selected parents and create offspring by combining their genetic information. This mimics the crossover of genetic material in natural reproduction.
4. **Mutation**: Introduce small random changes to the offspring's genetic information to promote diversity, similar to genetic mutations in nature.
5. **Evaluation**: Assess the fitness of the new population.
6. **Replacement**: Create a new population for the next generation by selecting individuals from the current population and the offspring.
7. **Termination**: Repeat the process for a predefined number of generations or until a satisfactory solution is found.

## Code:

from random import randint

def selection(li):

dec = list(map(lambda x : int(x, 2), li))

fit = list(map(lambda x : x\*x, dec))

s = sum(fit)

prob = list(map(lambda x : round(x/s, 3), fit))

avg = s/n

exe = list(map(lambda x : round(x/avg, 3), fit))

ac = list(map(lambda x : round(x), exe))

return dec, fit, prob, exe, ac

def pp(li, ac, n):

co = [] temp = [] index = []

for i in range(n):

if ac[i] == 1: co.append(li[i]) elif ac[i] >= 2:

for j in range(ac[i] - 1): temp.append(li[i]) co.append(li[i])

elif ac[i] == 0 and len(temp) != 0: co.append(temp[0]) temp.pop(0)

elif ac[i] == 0 and len(temp) == 0: index.append(i)

if len(index) != 0 and len(temp) != 0:

for i in index:

co.insert(i, temp[0]) temp.pop(0)

elif len(index) != 0 and len(temp) == 0:

co.insert(i, li[i]) return co

def cr(x):

s = 0

for i in x:

if i == '1':

s = s + 1 return s

def crossing(li, n):

crossed = []

for i in range(0, n, 2):

temp1 = li[i] j = i + 1 temp2 = li[j]

crosspoint = cr(temp1)

print("The crosspoint for pair " + str(i) + " is " + str(crosspoint)) temp3 = temp1[crosspoint: ]

temp4 = temp2[crosspoint: ]

temp1 = temp1[0 : crosspoint] + temp4 temp2 = temp2[0 : crosspoint] + temp3 crossed.append(temp1)

crossed.append(temp2) return crossed

def mutation(li, n):

mut = [] for i in li:

j = randint(0, n - 1)

print("For pair " + str(i) + ", the bit that will be changed is " + str(j)) if i[j] == '1':

i = i[0 : j] + '0' +i[j + 1 : ]

mut.append(i) elif i[j] == '0':

i = i[0 : j] + '1' +i[j + 1 : ]

mut.append(i) return mut

n = int(input("Enter number of samples: ")) sam = []

for i in range(n): sam.append(input("Enter gene: "))

m = int(input("Enter number of generations to be computed: ")) crossed = sam.copy()

for i in range(m):

dec, fit, prob, exe, ac = selection(crossed) s = sum(ac)

if s < n:

maxi = max(ac)

k = ac.index(maxi - 1) ac[k] += 1

if s > n:

maxi = max(ac)

k = ac.index(maxi) ac[k] -= 1

print("\n GENERATION ", i, "

")

print("Initial Population\tX Value\t\tFitness Value\tProbability\tExpected Count\t\tActual Count")

for j in range(n):

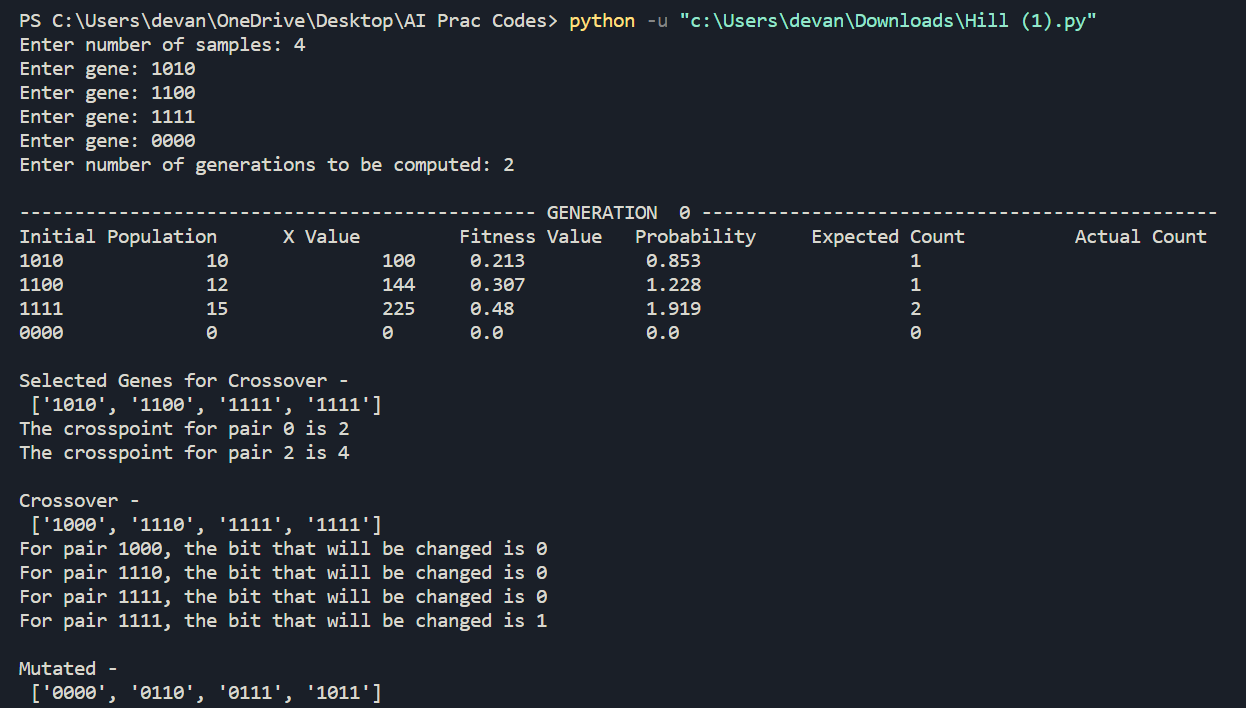
print(crossed[j], "\t\t", dec[j], "\t\t", fit[j], "\t", prob[j], "\t\t", exe[j], "\t\t\t", ac[j])

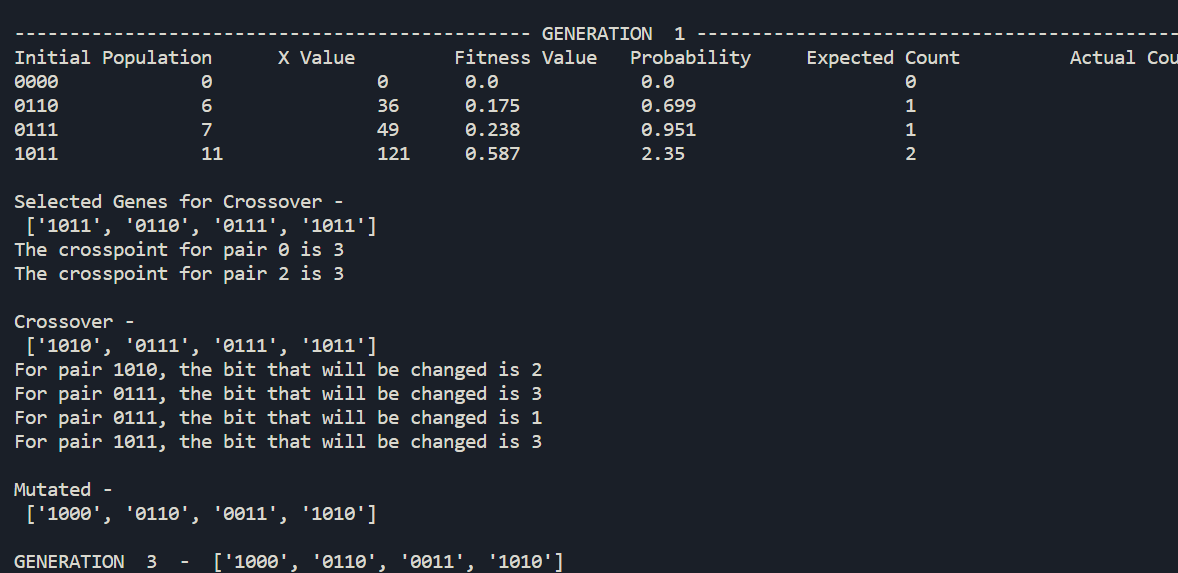
co = pp(crossed, ac, n)

print("\nSelected Genes for Crossover - \n", co) crossed = crossing(co, n)

print("\nCrossover - \n", crossed) crossed = mutation(crossed, n) print("\nMutated - \n", crossed)

print("\nGENERATION ", (m + 1), " - ", crossed)



****

**Conclusion:** In conclusion, Genetic algorithms efficiently explore solution spaces, demonstrating versatility and suitability for complex problems. However, their computational intensity, sensitivity to parameter choices, and the absence of optimality guarantees should be considered in their application.