Lab 1

Genetic Algorithm for Optimization Problems

2418/2025 LAB-IL molecus bayes								
							粉粉	
Genetic Al	gozithin				a soler			
5 main ph	ases. Ini	tializ"	, Fitness F	1551gnme	111,	n, Cabasove	70 (HQ3Q	
Tesminat"					The HALL Bearing			
except 2					assing and another			
Steps:	0 00 - 12	on to	Animus		TO STAN WITHTON			
iselecting encoding techniques					OF THE PARTY OF TH			
D to 31					: (panish probability next as			
2/5/201	initial po	a datio	on	: 150		and the con-		
-) Select	4	hore	70.00			Expected	Actual	
Staing	Initial	×	Fitness	Prob	o baop	count	count	
No.	popul" V	alue	100 = or,	EL CONTROL DE LA	12.67	Fallangesta	THE PERSON	
b:	01100	12	144	0.1247	40000	2.164	2	
2	11001	25	625	0.5411			-	
3	00101	5	25	0.0216	2-16	380.0	0	
Canada	10011	19	361 1155 A=288	0.3125	31.25	1.25	T. Carlo	
3)Select	mating I	oo deaq		of tenlesis			Jac Bi	
3) Select mating problems offspring after X fitness								
No.	pool	poto	t cass	sover	value	(x) = x2		
v	01100	4	01	101		P41		
2	11 001	in the	110	The second secon	24	576		
3 4	11001	2	110	20.0	21	129		
	10011		1 101	111111		P85		
4) (x0560	ves: Rando	im 4	- 2			esperatoria		
4) (xossover: Random 4.4.2 Max value 729								
simutation	VO.				Lineary, 1	trans and	ERRY TH	
Signed by	CSOSSOVE	23168	Nutat" Uhear			A X	tness	
1	10110		10000		nutat"	11000	30=2,	
2	11000		00000		000	The same of the same	841	
3	11011		00000		1000	4.7	576	
4	10001		10100		0100	Territoria de la constantina della constantina d	129	
	- No	1		4			400 546 4=636	

```
impost sandom
                                                    malicophi simi
def fitness (x):
    man places. Initialize, Filness Assignment soles to noute
                                            "tonime!
POPULATION-SIZE =4
CHROMOSOME_LENGTH = 5
MUTATION_RATE = 0.1
GENERATIONS = 10
def binary-to-decimal (binary):
   return int (binaxy, 2)
                                       neithbourg Loitini Loals
def decimal to binasy (n):
    seturn format (1, f'O {CHROMOSOME_LENGTH} b')
def initialize-population():
   seturn [decimal-to-binary (sandom. sandint (0, 2*+ LHROMOSOME-LENGITH)
                                    for in range (POPULATION_SIZE)
def evaluate population (population):
   seturn (fitness (binasy-to-decimal (individual)) for individual in population)
def select-parents (population, fitnesses):
   posents=[ ]
   fox in xange (2):
   is = 8andom. sample (sange(len (population)), 2)
   if fitnesses[i] > fitnesses[j]:
      parents append (population[i])
      posents append (population(j))
                                       Random 4 & 2
  return parents
olef crossavex (pasents, pasents):
   Point = sandom sandint (1, CHROMOSOME LENGITH -1)
   Childs = paxents [: point] + paxents [point:]
    Child2 = pasent2 [: point] + pasent1 [point:]
    setusn child, child2
```

```
def motate (individua):
   for bit in individual:
       if xandom. xandom() < MUTATION_RATE: { 1: "pol"
          mutated += 1' if bit == 0' else 0'
         mutated to bit bolomas: " - decides abdomed: " +
  seturn mutated wides abdust : " " at die abdust :
def genetic_algorithm(): (a) ... alamis alamid has also
   for generation in range (GENERATIONS):
      fitnesses = evaluate - population (population)
      new-population = [ ]
      while len (new-population) < POPULATION SIZET:
          paxent1, paxent2 = select-paxents (population, fitnesses)
          Child1, child2 = crossover (parent1, parent2)
          Child1 = mutate (child1)
    Child2 = mutate (child2)
          new-population. extend [[child1, child2]]
population = new_population [: POPULATION_SIZE]
     best = max (population, key = lambda x: fitness (binary-to-decimal
     print (f Generation [generation+1]: Best = foinary-to-decimal (best)],
                                                  : (a) ,600 (a))
              : Fitness = {fitness (binary to decimal (best))}')
  best max (population, key = lambda a: fitrtess (binary - to - decimal (au))
  print (Best solution: ", birary-to-decimal (best))
  point ("Fitness:", Fitness (binary-to-decimal (best)))
                                                · pais newbor
genetic-algorithm()
                                      det enal copy (sequilises):
```

```
Code:
import random
# 1. Define problem (fitness function)
def fitness(x):
  return x ** 2
# 2. Initialize parameters
POPULATION SIZE = 4
CHROMOSOME LENGTH = 5 # We'll use 5-bit binary strings, i.e. values 0–31
MUTATION RATE = 0.1
GENERATIONS = 10
# Helper: Binary to decimal
def binary to decimal(binary):
  return int(binary, 2)
# Helper: Decimal to binary
def decimal to binary(n):
  return format(n, f'0{CHROMOSOME LENGTH}b')
#3. Create initial population
definitialize population():
  return [decimal_to_binary(random.randint(0, 2**CHROMOSOME_LENGTH - 1)) for _ in
range(POPULATION_SIZE)]
```

4. Evaluate fitness of the population

```
def evaluate_population(population):
  return [fitness(binary_to_decimal(individual)) for individual in population]
# 5. Selection (Tournament Selection)
def select parents(population, fitnesses):
  parents = []
  for in range(2):
    i, j = random.sample(range(len(population)), 2)
    if fitnesses[i] > fitnesses[j]:
       parents.append(population[i])
     else:
       parents.append(population[j])
  return parents
# 6. Crossover (Single-point)
def crossover(parent1, parent2):
  point = random.randint(1, CHROMOSOME LENGTH - 1)
  child1 = parent1[:point] + parent2[point:]
  child2 = parent2[:point] + parent1[point:]
  return child1, child2
#7. Mutation (Bit-flip)
def mutate(individual):
  mutated = "
  for bit in individual:
     if random.random() < MUTATION RATE:
       mutated += '1' if bit == '0' else '0'
```

```
else:
       mutated += bit
  return mutated
#8. Iteration
def genetic algorithm():
  population = initialize population()
  for generation in range(GENERATIONS):
     fitnesses = evaluate_population(population)
    new_population = []
    while len(new population) < POPULATION SIZE:
       # Select
       parent1, parent2 = select_parents(population, fitnesses)
       # Crossover
       child1, child2 = crossover(parent1, parent2)
       # Mutation
       child1 = mutate(child1)
       child2 = mutate(child2)
       # Add to new population
       new population.extend([child1, child2])
    # Replace old population with new (trim if needed)
    population = new population[:POPULATION SIZE]
```

```
# Debug info
best = max(population, key=lambda x: fitness(binary_to_decimal(x)))
print(f''Generation {generation+1}: Best = {binary_to_decimal(best)}, Fitness =
{fitness(binary_to_decimal(best))}")

# 9. Output best solution
best = max(population, key=lambda x: fitness(binary_to_decimal(x)))
print("\nBest solution:", binary_to_decimal(best))
print("Fitness:", fitness(binary_to_decimal(best)))

# Run the genetic algorithm
genetic algorithm()
```

Output:

```
Shreya Raj 1BM23CS317
Generation 1: Best = 28, Fitness = 784
Generation 2: Best = 28, Fitness = 784
Generation 3: Best = 30, Fitness = 900
Generation 4: Best = 30, Fitness = 900
Generation 5: Best = 31, Fitness = 961
Generation 6: Best = 31, Fitness = 961
Generation 7: Best = 31, Fitness = 961
Generation 8: Best = 31, Fitness = 961
Generation 9: Best = 31, Fitness = 961
Generation 10: Best = 31, Fitness = 961
Best solution: 31
Fitness: 961
```