Om Shete - C3 - 2103163

Code: (Genetics*)

```
import random

def equation(a, b, c, d):
  return a + 2 * b + 17 * c + 14 * d


def eval_equation(individual):
  a, b, c, d = individual
  result = equation(a, b, c, d)
  return abs(result - 30)


def generate_individual(size):
  return [random.randint(0, 10) for _ in range(size)]


def crossover(parent1, parent2, crossover_prob):
  if random.random() < crossover_prob:
    crossover_point = random.randint(0, len(parent1) - 1)
    child = parent1[:crossover_point] + parent2[crossover_point:]
  else:
    child = parent1
  return child


def mutate(individual, mutation_rate):
  mutated_individual = individual[:]
  for i in range(len(mutated_individual)):
    if random.random() < mutation_rate:
      mutated_individual[i] = random.randint(0, 10)
  return mutated_individual


def genetic_algorithm(population_size, mutation_rate, crossover_prob):
  best_fitness = float('inf')
  best_individual = None
  generations = 0

  while True:
    generations += 1
```

```
    population = [generate_individual(4) for _ in
range(population_size)]

    for gen in range(population_size):
      fitnesses = [eval_equation(ind) for ind in population]

      min_fitness = min(fitnesses)
      best_index = fitnesses.index(min_fitness)
      best_individual = population[best_index]

      if min_fitness == 0:
        break

      selected = [random.choice(population) for _ in
range(population_size)]

      offspring = []
      for i in range(0, population_size, 2):
        parent1, parent2 = selected[i], selected[i + 1]
        child = crossover(parent1, parent2, crossover_prob)
        child = mutate(child, mutation_rate)
        offspring.extend([child])

      population[:] = offspring

    if min_fitness == 0 or generations >= 40:
      break

  print("Best individual:", best_individual)
  print("Fitness:", min_fitness)
  a, b, c, d = best_individual
  print("Population:", population_size)
  print("Solution: a={}, b={}, c={}, d={}".format(a, b, c, d))
  print("Equation result:", equation(a, b, c, d))
  print("Generations required:", generations)


if __name__ == "__main__":
  population_size = 10
  mutation_rate = 0.2
  crossover_prob = 0.5
  genetic_algorithm(population_size, mutation_rate, crossover_prob)
```

Output:

```
Run                                    Ask AI  503ms on 14:54:17, 04/05 ✓
Best individual: [2, 7, 0, 1]
Fitness: 0
Population: 10
Solution: a=2, b=7, c=0, d=1
Equation result: 30
Generations required: 25
```

Code:

```python
import random

def objective_function(solution):
  return sum(solution)

def generate_neighbor(current_solution):
  neighbor = current_solution[:]
  index = random.randint(0, len(neighbor) - 1)
  neighbor[index] = 1 - neighbor[index]
  return neighbor

def hill_climbing():
  current_solution = [random.randint(0, 1) for _ in range(10)]
  current_fitness = objective_function(current_solution)

  while True:
    neighbor = generate_neighbor(current_solution)
    neighbor_fitness = objective_function(neighbor)

    if neighbor_fitness >= current_fitness:
      current_solution = neighbor
      current_fitness = neighbor_fitness
    else:
      break

  return current_solution, current_fitness

best_solution, best_fitness = hill_climbing()
print("Best Solution:", best_solution)
print("Best Fitness:", best_fitness)
```

Output:

Run   Ask AI   444ms on 15:02:23, 04/05

```
Best Solution: [0, 1, 0, 0, 0, 1, 0, 1, 0, 0]
Best Fitness: 3
```