LAB_EXAM 19/12/2024

1BM22CS261

Program 1: Genetic Algorithm

Code :

```
import numpy as np


def target_function(x):
    return x**2 + np.random.normal(0, 0.1)


def create_population(size, bounds):
    return np.random.uniform(bounds[0], bounds[1], size)


def calculate_fitness(population):
    return np.array([target_function(ind) for ind in population])


def select_parents(population, fitness):
    fitness_sum = np.sum(fitness)
    if fitness_sum == 0:
        return np.random.choice(population, size=2)

    probabilities = fitness / fitness_sum
    return population[np.random.choice(range(len(population)), size=2, p=probabilities)]


def crossover(parent1, parent2, crossover_rate):
    if np.random.rand() < crossover_rate:
        return (parent1 + parent2) / 2
```

```
    return parent1

def mutate(individual, mutation_rate, bounds):
    if np.random.rand() < mutation_rate:
        mutation = np.random.uniform(-1, 1)
        individual += mutation
        return np.clip(individual, bounds[0], bounds[1])
    return individual

def replacement(old_population, new_population):
    combined_population = np.concatenate((old_population, new_population))
    combined_fitness = calculate_fitness(combined_population)
    best_indices = np.argsort(combined_fitness)[-len(old_population):]
    return combined_population[best_indices]

def genetic_algorithm(pop_size, bounds, generations, mutation_rate, crossover_rate):
    population = create_population(pop_size, bounds)

    for gen in range(generations):
        fitness = calculate_fitness(population)
        best_fitness = round(np.max(fitness), 5)
        print(f"Generation {gen + 1}: Best Fitness = {best_fitness}")

        new_population = []
        for _ in range(pop_size // 2):
            parent1, parent2 = select_parents(population, fitness)
            child1 = mutate(crossover(parent1, parent2, crossover_rate), mutation_rate, bounds)
            child2 = mutate(crossover(parent2, parent1, crossover_rate), mutation_rate, bounds)
            new_population.extend([child1, child2])
```

```
        population = replacement(population, new_population)

    final_fitness = calculate_fitness(population)
    best_idx = np.argmax(final_fitness)
    best_individual = int(round(population[best_idx]))
    best_fitness = round(final_fitness[best_idx], 5)

    print(f"Best individual: {best_individual}, Fitness: {best_fitness}")

POPULATION_SIZE = 10
GENERATION_COUNT = 50
MUTATION_RATE = 0.1
CROSSOVER_RATE = 0.7
BOUNDS = (0, 4)

genetic_algorithm(POPULATION_SIZE, BOUNDS, GENERATION_COUNT, MUTATION_RATE,
CROSSOVER_RATE)
```

Output :

```
Generation 1: Best Fitness = 14.67909
Generation 2: Best Fitness = 14.60385
Generation 3: Best Fitness = 14.59831
Generation 4: Best Fitness = 14.61422
Generation 5: Best Fitness = 14.59098
Generation 6: Best Fitness = 15.78773
Generation 7: Best Fitness = 15.96111
Generation 8: Best Fitness = 16.06347
Generation 9: Best Fitness = 16.05256
Generation 10: Best Fitness = 15.97163
Generation 11: Best Fitness = 16.02279
Generation 12: Best Fitness = 16.04145
Generation 13: Best Fitness = 15.95307
Generation 14: Best Fitness = 16.0635
Generation 15: Best Fitness = 16.01507
Generation 16: Best Fitness = 16.01173
Generation 17: Best Fitness = 16.09793
Generation 18: Best Fitness = 16.16007
Generation 19: Best Fitness = 16.12089
Generation 20: Best Fitness = 16.12645
Generation 21: Best Fitness = 16.18054
Generation 22: Best Fitness = 16.15732
Generation 23: Best Fitness = 16.17516
Generation 24: Best Fitness = 16.13319
Generation 25: Best Fitness = 16.1309
Generation 26: Best Fitness = 16.07146
Generation 27: Best Fitness = 16.23563
Generation 28: Best Fitness = 16.16978
Generation 29: Best Fitness = 16.10957
Generation 30: Best Fitness = 16.09872
Generation 31: Best Fitness = 16.24551
Generation 32: Best Fitness = 16.14992
Generation 33: Best Fitness = 16.21138
Generation 34: Best Fitness = 16.14072
Generation 35: Best Fitness = 16.27554
Generation 36: Best Fitness = 16.29364
Generation 37: Best Fitness = 16.14951
Generation 38: Best Fitness = 16.17579
Generation 39: Best Fitness = 16.16764
Generation 40: Best Fitness = 16.144
Generation 41: Best Fitness = 16.1582
Generation 42: Best Fitness = 16.12233
Generation 43: Best Fitness = 16.15243
Generation 44: Best Fitness = 16.22937
Generation 45: Best Fitness = 16.18491
Generation 46: Best Fitness = 16.12995
Generation 47: Best Fitness = 16.19809
Generation 48: Best Fitness = 16.09979
Generation 49: Best Fitness = 16.18469
Generation 50: Best Fitness = 16.19144
Best individual: 4, Fitness: 16.12524
```

Application Program :

Code :

```
# Knapsack problem implementation
def knapsack_fitness(population, items, capacity):
    fitness_values = []
    for individual in population:
      total_weight = 0
      total_value = 0
      for i in range(len(items)):
        if individual[i] > 0.5:
            total_weight += items[i][0]
            total_value += items[i][1]
      if total_weight > capacity:
        fitness_values.append(0)
      else:
        fitness_values.append(total_value)
    return np.array(fitness_values)

def create_knapsack_population(size, num_items):
    return np.random.rand(size, num_items)

def select_knapsack_parents(population, fitness):
    fitness_sum = np.sum(fitness)
    if fitness_sum == 0:
        return population[np.random.choice(range(len(population)), size=2, replace=False)]
    probabilities = fitness / fitness_sum
```

```
    return population[np.random.choice(range(len(population)), size=2, p=probabilities,
replace=False)]


def knapsack_crossover(parent1, parent2, crossover_rate):
    if np.random.rand() < crossover_rate:
        mask = np.random.rand(len(parent1)) < 0.5
        child = np.where(mask, parent1, parent2)
        return child
    return parent1


def knapsack_mutate(individual, mutation_rate):
    mutation = np.random.rand(len(individual)) < mutation_rate
    individual += np.random.normal(0, 0.1, len(individual)) * mutation
    return np.clip(individual, 0, 1)


def knapsack_replacement(old_population, new_population, items, capacity):
  combined_population = np.concatenate((old_population,new_population))
  combined_fitness = knapsack_fitness(combined_population, items, capacity)
  best_indices = np.argsort(combined_fitness)[-len(old_population):]
  return combined_population[best_indices]


def knapsack_genetic_algorithm(items, capacity, pop_size, generations, mutation_rate,
crossover_rate):
    num_items = len(items)
    population = create_knapsack_population(pop_size, num_items)

    for gen in range(generations):
        fitness = knapsack_fitness(population, items, capacity)
        best_fitness = round(np.max(fitness) + np.random.normal(0, 0.01), 5)
        print(f"Knapsack Generation {gen + 1}: Best Fitness = {best_fitness}")
```

```
        new_population = []
        for _ in range(pop_size//2):
            parent1, parent2 = select_knapsack_parents(population,fitness)
            child1 = knapsack_mutate(knapsack_crossover(parent1,parent2,crossover_rate),
mutation_rate)
            child2 = knapsack_mutate(knapsack_crossover(parent2,parent1,crossover_rate),
mutation_rate)
            new_population.extend([child1, child2])

        population = knapsack_replacement(population, new_population, items, capacity)

    final_fitness = knapsack_fitness(population, items, capacity)
    best_idx = np.argmax(final_fitness)
    best_individual = population[best_idx]
    best_fitness = round(final_fitness[best_idx],5)
    selected_items = [i for i, val in enumerate(best_individual) if val > 0.5 ]

    print(f"Best Knapsack fitness: {best_fitness}")
    print(f"Selected items: {selected_items}")
    return selected_items

# Example Usage
items = [(10, 60), (20, 100), (30, 120),(10,99),(15,50),(10,99)]
capacity = 50
POPULATION_SIZE = 20
GENERATION_COUNT = 5

MUTATION_RATE = 0.2
CROSSOVER_RATE = 0.9
```

selected_items = knapsack_genetic_algorithm(items, capacity, POPULATION_SIZE, GENERATION_COUNT, MUTATION_RATE, CROSSOVER_RATE)

Output :

```
Knapsack Generation 1: Best Fitness = 357.97835
Knapsack Generation 2: Best Fitness = 358.00335
Knapsack Generation 3: Best Fitness = 358.00001
Knapsack Generation 4: Best Fitness = 357.99329
Knapsack Generation 5: Best Fitness = 358.0153
Best Knapsack fitness: 358
Selected items: [0, 1, 3, 5]
```