1BM22CS241

Sanjeet Prajwal Pandit

Gene_Expression

```
import numpy as np
import random

# Define the problem: Optimization function (e.g., Sphere function)
def fitness_function(x):
    return -sum(x_i ** 2 for x_i in x)  # Minimize the negative of the
Sphere function

# Parameters
population_size = 50
num_genes = 5
mutation_rate = 0.1
crossover_rate = 0.7
num_generations = 100
search_space = (-10, 10)

# Initialize Population
def initialize_population(pop_size, num_genes, search_space):
    return np.random.uniform(search_space[0], search_space[1], (pop_size,
num_genes))

# Evaluate Fitness
def evaluate_fitness(population):
    return np.array([fitness_function(individual) for individual in
population])

# Selection: Tournament Selection
def tournament_selection(population, fitness):
    selected = []
    for _ in range(len(population)):
        i, j = random.sample(range(len(population)), 2)
        selected.append(population[i] if fitness[i] > fitness[j] else
population[j])
    return np.array(selected)

# Crossover: Single-point crossover
def crossover(parent1, parent2, rate):
    if random.random() < rate:
        point = random.randint(1, len(parent1) - 1)
        child1 = np.concatenate((parent1[:point], parent2[point:]))
        child2 = np.concatenate((parent2[:point], parent1[point:]))
        return child1, child2
    return parent1, parent2

# Mutation: Gaussian mutation
def mutate(individual, rate, search_space):
    for i in range(len(individual)):
        if random.random() < rate:
            individual[i] += np.random.normal(0, 1)
            individual[i] = np.clip(individual[i], search_space[0],
search_space[1])
    return individual

# Gene Expression: Here it's a direct mapping
```

```python
# (In more complex cases, this could involve encoding/decoding operations)
def gene_expression(individual):
    return individual

# Main GEA Loop
population = initialize_population(population_size, num_genes,
search_space)

for generation in range(num_generations):
    fitness = evaluate_fitness(population)

    # Track the best solution
    best_idx = np.argmax(fitness)
    best_solution = population[best_idx]
    best_fitness = fitness[best_idx]

    print(f"Generation {generation + 1}: Best Fitness = {best_fitness}")

    # Selection
    selected_population = tournament_selection(population, fitness)

    # Crossover
    next_generation = []
    for i in range(0, len(selected_population), 2):
        parent1 = selected_population[i]
        parent2 = selected_population[i + 1 if i + 1 <
len(selected_population) else 0]
        child1, child2 = crossover(parent1, parent2, crossover_rate)
        next_generation.append(child1)
        next_generation.append(child2)

    # Mutation
    next_generation = [mutate(ind, mutation_rate, search_space) for ind in
next_generation]

    # Gene Expression (if necessary)
    population = np.array([gene_expression(ind) for ind in
next_generation])

# Output the best solution
print(f"Best Solution Found: {best_solution}")
print(f"Best Fitness: {best_fitness}")
```

```
Best Solution Found: [ 0.04025503 -0.00115442  0.07148469  0.00157052  0.03
007435]
Best Fitness: -0.007638793344910918
```