Experiment No. 7

Title - Implement a program of Genetic Algorithm System

Theory -

Genetic Algorithms (GAs) are heuristic optimization techniques inspired by the principles of natural selection and genetics. They simulate the process of natural selection to find optimal or near-optimal solutions to problems that might have multiple possible solutions. GAs are particularly useful for complex search spaces and optimization problems where traditional methods might struggle to find the best solution.

Working of Genetic Algorithms

- Initialization:

A population of potential solutions (often called individuals, chromosomes, or genotypes) is randomly created.

- Evaluation (Fitness):

Each individual's fitness is assessed using a fitness function that quantifies how well it solves the problem.

- Selection:

Individuals are selected for reproduction based on their fitness. Higher fitness individuals have a higher chance of being selected. Various selection methods like roulette wheel selection, tournament selection, or rank-based selection are used.

- Reproduction (Genetic Operations):

Selected individuals are combined using genetic operators:

  - Crossover (Recombination): Genetic material is exchanged between selected individuals to create new solutions.

- Mutation: Introducing random changes in the genetic information helps explore new areas of the search space.

- Replacement:

The new offspring population replaces the old population through methods like generational replacement or elitism (keeping the best individuals).

- Termination:
- The process continues iteratively until a termination condition is met, such as reaching a maximum number of generations or finding a satisfactory solution.

- **Steps of a Genetic Algorithm**

1. Initialization

Objective : Generate an initial population of potential solutions.

Process :

- Create a set of individuals (chromosomes or solutions) randomly or through some heuristic methods.
- These individuals form the initial population and represent potential solutions to the problem being optimized.
- The population size, representation scheme, and initial values are critical for the algorithm's performance.

2. Fitness Evaluation

Objective : Assess the quality of each individual in the population.

Process :

- Utilize a fitness function that quantifies how well an individual solves the problem.
- The fitness function can be problem-specific and measures how close an individual is to the optimal solution.
- Individuals with higher fitness scores are considered better solutions

3. Selection

Objective : Choose individuals from the population for reproduction based on their fitness.

Process :

- Higher fitness individuals have a higher probability of being selected for reproduction.
- Selection methods like roulette wheel selection, tournament selection, or rank-based selection are used.
- The aim is to create a mating pool with individuals that contribute positively to the next generation.

4. Crossover and Mutation

Objective : Create new solutions by combining genetic material from selected individuals.

Process :

- Crossover (Recombination): Exchange genetic information between selected parents to produce offspring.
- Single-point, multi-point, or uniform crossover methods are common.
- Mutation: Introduce small random changes in some individuals to maintain diversity.
- Randomly modify specific genes within individuals to explore new areas of the search space.
- Mutation helps prevent premature convergence and aids in exploring the solution space.

5. Replacement

Objective : Replace the old population with a new generation of individuals.

Process :

- Determine how the new generation replaces the old one.
- Common methods include generational replacement (entirely replacing the population) or elitism (keeping the best individuals from the previous generation).

6. Termination

Objective : Decide when to stop the algorithm.

Process :

- Termination conditions can include reaching a maximum number of generations, achieving a satisfactory solution, or no significant improvement over successive iterations.
- Once the termination condition is met, the algorithm stops, and the best solution found so far is considered the output.

Implementation :

1. Importing Libraries

```
import random
import numpy as np
```

- `random`: Used for generating random numbers.
- `numpy`: Used for numerical operations, particularly here for square root (`np.sqrt`) in the fitness function.

2. Fitness Function

```
def calculate_fitness(individual):
    x = individual[0]
    y = individual[1]
    distance = np.sqrt(x**2 + y**2)
    return abs(distance - 1.0)
```

- `calculate_fitness`: Evaluates the fitness (closeness to pi) of an individual by calculating its distance from the point (0, 0) to approximate the unit circle. It measures the absolute difference between the calculated distance and 1.0, representing the radius of the unit circle.

3. Population Initialization

```
def generate_population(size):
    return [[random.uniform(-1, 1), random.uniform(-1, 1)] for _ in range(size)]
```

- `generate_population`: Creates an initial population of individuals by generating random (x, y) coordinates within the range [-1, 1] for a specified population size.

4. Crossover Operation

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

- `crossover`: Combines genetic material from two parents to create a child individual. It selects a crossover point and combines parts of the genetic information from both parents.

5. Mutation Operation

```
def mutate(individual, mutation_rate):
  for i in range(len(individual)):
    if random.random() < mutation_rate:
      individual[i] = random.uniform(-1, 1)
  return individual
```

- `mutate`: Introduces random changes in an individual's genes with a given probability (`mutation_rate`). It randomly selects genes and modifies them within the range [-1, 1].

6. Genetic Algorithm Implementation

```
def approximate_pi(population_size, generations, mutation_rate):
  population = generate_population(population_size)
    best_approximation = [float('inf'), None]  # [Best Fitness, Individual] -> best fitness assigned as infinity at first

  for generation in range(generations):
    fitness_scores = [calculate_fitness(individual) for individual in population]

    # Find the individual with the lowest fitness (closest to pi)
    best_index = fitness_scores.index(min(fitness_scores))
    best_individual = population[best_index]
    best_fitness = fitness_scores[best_index]

    # Update best approximation if a better one is found
    if best_fitness < best_approximation[0]:
      best_approximation = [best_fitness, best_individual, generation]

    # Output the best approximation in each generation
    print(f"Generation {generation}: Best approximation to pi = {np.pi - best_fitness}")

    # Create the next generation using crossover and mutation
    new_population = []
    for _ in range(population_size):
```

```
        parent1 = random.choice(population)
        parent2 = random.choice(population)
        child = crossover(parent1, parent2)
        child = mutate(child, mutation_rate)
        new_population.append(child)

    population = new_population

return best_approximation
```

- `approximate_pi`: The main function that executes the Genetic Algorithm. It initializes the population, iterates through generations, performs selection, crossover, mutation, and updates the best approximation to pi found so far.

7. Parameter Definitions and Execution

```
population_size = 1000
generations = 10000
mutation_rate = 0.1

best_pi_info = approximate_pi(population_size, generations, mutation_rate)
best_pi_fitness, best_pi_individual, best_pi_generation = best_pi_info
print(f"Best approximation of pi ({np.pi - best_pi_fitness}) found at generation {best_pi_generation}")
```

- `population_size`, `generations`, and `mutation_rate` are defined.
- The Genetic Algorithm is executed with these parameters, and the best approximation of pi and the generation it was found are printed at the end.

This implementation attempts to approximate the value of pi by evolving a population of points closer to the unit circle using a Genetic Algorithm. It employs selection, crossover, and mutation operations on a population of individuals to iteratively improve the approximation to pi.

Usage of Genetic Algorithms

- ***GAs find applications in various fields***

- Optimization Problems: Traveling Salesman Problem, Knapsack Problem, etc.
- Machine Learning: Feature Selection, Neural Network Training, etc.
- Engineering: Design Optimization, Scheduling Problems, etc.
- Financial Modelling: Portfolio Optimization, Forecasting, etc.

- **Advantages of Genetic Algorithms**

- Exploration and Exploitation: GAs balance exploration of new solutions and exploitation of known solutions.
- Adaptability: Effective for complex and diverse problem spaces.
- No Gradient Requirement: Suitable for problems with non-linear, discontinuous, or noisy search spaces.
- Parallelism: Easily parallelisable, allowing for efficient implementation on parallel architectures.

Limitations of Genetic Algorithms

- Performance: May require a large number of evaluations, making them computationally expensive.
- No Guarantee of Global Optimum: GAs provide good solutions but not necessarily the best or optimal solution.
- Parameter Tuning: Performance may heavily depend on parameter settings.
- Problem Representation: Effective representation of the problem domain can be challenging.

Summary

Genetic Algorithms mimic the process of natural selection to find optimal or near-optimal solutions to complex problems. They involve initializing a population, evaluating fitness, selecting individuals for reproduction, using genetic operators for reproduction, replacing the population, and iterating until a termination condition is met. GAs have broad applications, offer adaptability to various problem domains, but come with computational costs and no guarantee of finding the global optimum. Their effectiveness relies on problem representation, parameter settings, and appropriate design choices.