Dr. Vishwanath Karad
MIT WORLD PEACE UNIVERSITY | PUNE
TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

Final Year BTech. (EE)

Semester: 5 **Subject: AIML**

Name: Shreerang Mhatre **Class: TY**
Roll No.: 52 **Batch: A3**

Experiment No. 7

Aim: Implementation of Simple Genetic Algorithm

Objective:

To get familiarize with Mathematical foundations for Genetic algorithm, operator.
To study the Applications of Genetic Algorithms

Software Required:

MATLAB

Theory:

Genetic algorithm is a search technique used in computing to find true or approximate solutions to approximate solutions to optimization & search problems.

Genetic algorithms are inspired by Darwin's theory about evolution. Solution to a problem solved by genetic algorithms is evolved.

Algorithm is started with a set of solutions (represented by chromosomes) called population. Solutions from one population are taken and used to form a new population.

This is motivated by a hope, that the new population will be better than the old one. Solutions which are selected to form new solutions (offspring) are selected according to their fitness - the more suitable they are the more chances they have to reproduce. This is repeated until some condition (for example number of populations or improvement of the best solution) is satisfied

Outline of the Basic Genetic Algorithm

1. **[Start]** Generate random population of *n* chromosomes (suitable solutions for the problem)

2. **[Fitness]** Evaluate the fitness *f(x)* of each chromosome *x* in the population

3. **[New population]** Create a new population by repeating following steps until the new population is complete

1. **[Selection]** Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected)

2. **[Crossover]** With a crossover probability cross over the parents to form a new offspring (children). If no crossover was performed, offspring is an exact copy of parents.

3. **[Mutation]** With a mutation probability mutate new offspring at each locus (position in chromosome).

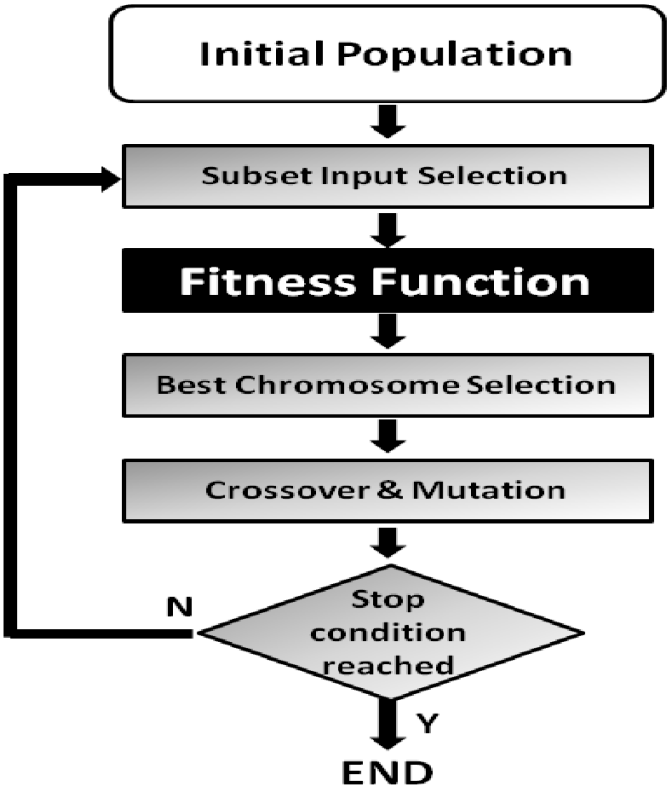
4. [**Accepting**] Place new offspring in a new population

5. [**Replace**] Use new generated population for a further run of algorithm

6. [**Test**] If the end condition is satisfied, stop, and return the best solution in current population

7. [**Loop**] Go to step 2

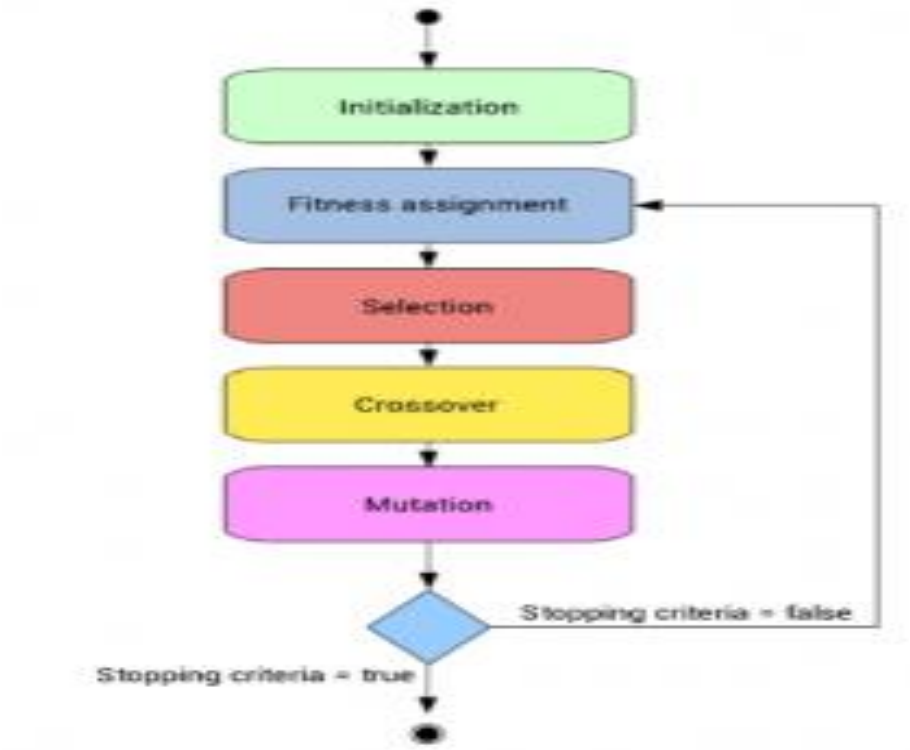
Flowchart



Algorithm:

- **Firstly, we defined our initial population as our countrymen.**
- **We defined a function to classify whether is a person is good or bad.**
- **Then we selected good people for mating to produce their off-springs.**
- **And finally, these off-springs replace the bad people from the population and this process repeats.**
- **This is how genetic algorithm actually works, which basically tries to mimic the human evolution to some extent.**
- **it is an optimization technique, which tries to find out such values of input so that we get the best output values or results.**

The working of a genetic algorithm is also derived from biology



Pseudo code for GA:

START

Generate the initial population

Compute fitness

REPEAT

Selection

Crossover

Mutation

Compute fitness

UNTIL population has converged

STOP

Conclusion:

Post Lab Questions:

1. Name some of the existing search methods.
2. What are the operators involved in a simple genetic algorithm?
3. What is reproduction?
4. What is crossover?
5. Write the code for GA and implement it using Python.

CODE:

```
import random

def fitness_function(individual):
    return sum(individual)

def generate_individual():
    return [random.randint(0, 100) for _ in range(10)]

def mutate(individual):
    index_to_mutate = random.randint(0, len(individual) - 1)
    individual[index_to_mutate] = random.randint(0, 100)
    return individual

def crossover(parent1, parent2):
    midpoint = len(parent1) // 2
    child1 = parent1[:midpoint] + parent2[midpoint:]
    child2 = parent2[:midpoint] + parent1[midpoint:]
    return child1, child2

def select_parents(population):
    fitness_values = [fitness_function(ind) for ind in population]
    total_fitness = sum(fitness_values)
    probabilities = [fit / total_fitness for fit in fitness_values]
    parents = random.choices(population, probabilities, k=2)
    return parents

def genetic_algorithm():
    population_size = 100
    population = [generate_individual() for _ in range(population_size)]

    for generation in range(100):
        parents = select_parents(population)
        offspring = []

        for i in range(population_size // 2):
            parent1, parent2 = parents
            child1, child2 = crossover(parent1, parent2)
            offspring.append(mutate(child1))
            offspring.append(mutate(child2))

        population = parents + offspring
        population.sort(key=fitness_function, reverse=True)
        population = population[:population_size]

    return population[0]
```

```
best_individual = genetic_algorithm()
```

```
print("Best Individual:", best_individual)
print("Fitness Value:", fitness_function(best_individual))
```

```
Best Individual: [25, 17, 38, 13, 51, 38, 95, 79, 99, 99]
Fitness Value: 554
```

Exp-7 Implementation of simple Genetic Algorithm

PAGE No.
DATE 24/10/23

* Post Lab Questions:

Q1) Name some of the existing search methods.

> Some search methods are -

1. Linear search
2. Binary search
3. Depth-First search (DFS)
4. Breadth-First search (BFS)
5. A* search
6. Greedy search
7. Hill Climbing
8. Genetic Algorithms
9. Simulated Annealing
10. Beam search
11. Best-First-search
12. Tabu search
13. Particle swarm optimization (PSO)
14. Ant colony optimization (ACO)
15. Dijkstra's Algorithm
16. Floyd-Warshall Algorithm.

Q2) What are the operators involved in a simple genetic algorithm?

-> The main operators involved in a simple genetic algorithm are -

① Initialization -
In first generation, a population of candidate solutions is randomly generated

② Selection -
Selection is the process of choosing individuals from current population to become parents for the next generation

③ Crossover -
Crossover involves taking two or more parent individuals and combining their genetic information to create one or more offspring.

④ Mutation -
A small random change applied to an individual's genetic information.

⑤ Evaluation -
Each individual's fitness is evaluated to determine how well it solves the problem.

Q3) what is reproduction?

-> Reproduction is a crucial genetic operator in genetic algorithms & evolutionary computation. It involves selecting indiviuals from a population based on their fitness, serving as parents for the next generation. These selected individuals undergo crossover to combine their genetic material, introducing diversity and potentially benefitials traits. In some cases, mutation is applied to farther diversify the offspring.

Q4) what is crossover?

-> In genetic Algorithms & evolutionary computation, "crossover" refers to a genetic operator that combines genetic information from two or more parent individuals to produce one or more offspring. This operation is applied during the reproduction phase, and it mimics the process of recombination or mating in biological genetics crossover is a key mechanism for introducing genetic diversity & potentially combining beneficial traits from the parents, contributing to the evolution of solution.

Q5) Write the code GA & implement it using Python.

->
```
import random
# Define target function to be maximized
def fitness_function(x):
    return x**2
# GA Parameters
population_size = 100
mutation_rate = 0.1
generation = 100
# Initialize the population with
  random individuals
def initialize_population(size):
    return [random.uniform(-10,10)
        for _ in range(size)]
# Select two individuals with a probability
  based on their fitness
def select_parents(population):
    return random.choice(population, k=2,
      weights = [fitness_function(x) for
      x in population])
# Perform single-point crossover
def crossover(parent1, parent2):
    crossover_point = random.randint
        (0, len(parent)-1)
    child1 = parent1[:crossover_point] +
            parent2[crossover_point:]
    child2 = parent2[:crossover_point] +
            parent1[crossover_point:]
    return child1, child2
```

```
# Apply mutation with a probability for
  each gene
def mutate (individual-rate):
    return [gene + random.uniform)
# Main GA loop
population = initialize-population(population-
    size)
for generation in range (generations):
    next_population = []

    for _ in range (population-size // 2):
        parent1, parent2 = select_parents (population)
        child1, child2 = crossover (parent1, parent2)
        child1 = mutate (child1, mutation_rate)
        child2 = mutate (child2, mutation-rate)
        next_population.extend ([child1, child2])

    population = next_population.
# Find the individual with the
  highest fitness
best_individual = max (population,
                key = fitness_function)
best_fitness = function (best_individual)

print (f" Best individual: {best_individual}")
print (f" Best fitness: {best_fitness}").
```