1. Genetic Algorithm for optimization of travelling salesman problem

Lab - 1

Genetic Algorithm for Optimization Problems 11/8/25.

Genetic Algorithm is an optimization technique inspired by natural selection.
It works by evolving a population of candidate solutions to improve their quality over time
We first define the problem & the fitness function that measures how good each solution is.
Initial population is generated randomly
In each generation, selection chooses the best individual to reproduce
Crossover mixes of two solutions to create new ones
Mutation introduces small random changes to maintain diversity & avoid getting stuck.
This process repeats for a set of generations until best solution stabilizes

Applications.

* We widely used when the search space is large, complex or non-linear.
* Scheduling, route optimization, engineering design, game AI, machine learning parameter tuning

Genetic Algorithm Application – Travelling Salesman Problem (TSP)

The travelling salesman problem is one of the most famous optimization problems. A salesman must visit a given set of cities exactly once & return to the starting city while minimizing the total travel distance

- TSP is NP hard – no efficient exact algorithm exists for large no of city
- Brute force could require checking all possible permutations of routes (which is factorial in complexity)
- Genetic Algorithm (GA) provide an effective way to find a near optimal solution in reasonable time
- Widely used in logistics (Amazon, Swiggy, airline scheduling)

→ Why GA is best here:

Other algorithms (like dynamic programming, branch & bound) become impractical when cities are large (50, 100, 1K+). GA scales well & gives close to optimal solutions quickly.

Pseudocode:

Input: Set of cities & distance matrix

1. Initialize population:
   - Generate random routes (permutation of cities)
2. Evaluate fitness of each route:
   - Fitness = $\frac{1}{\text{total distance of route}}$
3. Repeat until max generations:
   a. Selection:
      * Select parents based on fitness

b. Crossover :
- Apply order crossover (OX) to generate children routes
c. Mutation:
- Swap two cities in the route randomly.
d. Evaluation:
- Compute total distance & fitness for new routes
e. Replacement:
- Form new population (elitism keeps best route)
4. Return the best route & its total distance

Output:

Number of cities: 10
City coordinates : [(16,1), (65,96), (30,42), (58,100), (40,67), (40,25), (67,7), (64,64), (49,56), (16,18)]
Initialization cities & population ...
Starting Genetic Algorithm ...
Generation 1: Best Distance = 305.39
Generation 2: Best Distance = 305.39
Generation 3: Best Distance = 279.97
Generation 4: Best Distance = 279.97
Generation 5: Best Distance = 279.97
Generation 6: Best Distance = 279.97
No improvement for 3 generations. Stopping early
Best Route Found: [9,0,6,5,7,1,3,8,4,2]
Best Distance: 279.97

Python code:

```
import random
import math

# Calculate total distance of a route
def route_distance(route, cities):
    dist = 0
    for i in range(len(route)):
        x1, y1 = cities[route[i]]
        x2, y2 = cities[route[(i + 1) % len(route)]]
        dist += math.sqrt((x1 - x2)**2 + (y1 - y2)**2)
    return dist

# Create initial population
def initial_population(pop_size, city_count):
    population = []
    for _ in range(pop_size):
        route = list(range(city_count))
```

```
        random.shuffle(route)
        population.append(route)
    return population

# Tournament selection
def selection(population, cities):
    selected = random.sample(population, 3)
    selected.sort(key=lambda r: route_distance(r, cities))
    return selected[0]

# Order crossover
def crossover(parent1, parent2):
    start, end = sorted(random.sample(range(len(parent1)), 2))
    child = [None] * len(parent1)
    child[start:end] = parent1[start:end]
    pointer = 0
    for gene in parent2:
        if gene not in child:
            while child[pointer] is not None:
                pointer += 1
            child[pointer] = gene
    return child

# Mutation (swap two cities)
def mutate(route, mutation_rate=0.1):
    if random.random() < mutation_rate:
        i, j = random.sample(range(len(route)), 2)
        route[i], route[j] = route[j], route[i]
    return route

# Genetic Algorithm with early stopping
def genetic_algorithm(cities, pop_size=50, generations=100):
    print("Initializing cities and population...")
    population = initial_population(pop_size, len(cities))

    best_route = min(population, key=lambda r: route_distance(r, cities))
    best_distance = route_distance(best_route, cities)

    print("Starting Genetic Algorithm...\n")
```

```
    no_improvement_count = 0

    for gen in range(1, generations + 1):
        new_population = []
        for _ in range(pop_size):
            parent1 = selection(population, cities)
            parent2 = selection(population, cities)
            child = crossover(parent1, parent2)
            child = mutate(child)
            new_population.append(child)

        population = new_population
        current_best = min(population, key=lambda r: route_distance(r, cities))
        current_distance = route_distance(current_best, cities)

        if current_distance < best_distance:
            best_route, best_distance = current_best, current_distance
            no_improvement_count = 0
        else:
            no_improvement_count += 1

        print(f"Generation {gen}: Best Distance = {best_distance:.2f}")

        # Stop if no improvement for 3 consecutive generations
        if no_improvement_count >= 3:
            print("\nNo improvement for 3 generations. Stopping early.")
            break

    return best_route, best_distance

# Example run
if __name__ == "__main__":
    city_count = 10
    cities = [(random.randint(0, 100), random.randint(0, 100)) for _ in
range(city_count)]

    print(f"Number of Cities: {city_count}")
    print("City Coordinates:", cities, "\n")

    best_route, best_distance = genetic_algorithm(cities)
```

```
print("\nBest Route Found:", best_route)
print("Best Distance:", best_distance)
```

Output:

Output

Clear

```
Number of Cities: 10
City Coordinates: [(16, 1), (65, 96), (30, 42), (58, 100), (40, 67),
    (70, 25), (67, 7), (64, 64), (49, 56), (16, 18)]

Initializing cities and population...
Starting Genetic Algorithm...

Generation 1: Best Distance = 305.39
Generation 2: Best Distance = 305.39
Generation 3: Best Distance = 279.97
Generation 4: Best Distance = 279.97
Generation 5: Best Distance = 279.97
Generation 6: Best Distance = 279.97

No improvement for 3 generations. Stopping early.

Best Route Found: [9, 0, 6, 5, 7, 1, 3, 8, 4, 2]
Best Distance: 279.97113971002153

=== Code Execution Successful ===
```