Question:9

TRAVELING SALESMAN USING GENETIC ALGORITHM

AIM

To solve Traveling Salesman problem by implementing genetic algorithm using Python

ALGORITHM

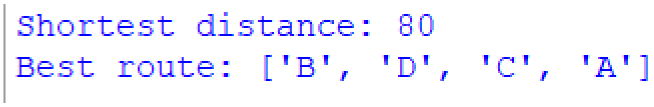
1. **Calculate Total Distance:**
   - Define a function `calculate_total_distance(route, distances)` to calculate the total distance of a given route.
2. **Generate Initial Population:**
   - Define a function `generate_initial_population(num_cities, population_size)` to generate an initial population of random routes.
3. **Crossover:**
   - Define a function `crossover(parent1, parent2)` to perform crossover between two parent routes.
4. **Mutation:**
   - Define a function `mutate(route, mutation_rate)` to introduce mutations in a route with a given mutation rate.
5. **Genetic Algorithm:**
   - Define a function `genetic_algorithm(cities, distances, population_size=100, generations=100, mutation_rate=0.01)` to evolve the population over a certain number of generations.
   - Initialize the population with random routes.
   - Iterate over a specified number of generations:
     - Perform selection, crossover, and mutation to generate a new population.
   - Return the best route found.
6. **Example Usage**:
   - Define a list of cities and distances between them.
   - Call the `genetic_algorithm` function to find the shortest route and its distance.
   - Print the results.

CODE

```
import random
import itertools
def calculate_total_distance(route, distances):
    return sum(distances[route[i]][route[(i + 1) % len(route)]] for i in range(len(route)))
```

```
def generate_initial_population(num_cities, population_size):
    return [list(range(num_cities)) for _ in range(population_size)]

def crossover(parent1, parent2):
    crossover_point = random.randint(0, len(parent1) - 1)
    return parent1[:crossover_point] + [city for city in parent2 if city not in
parent1[:crossover_point]]

def mutate(route, mutation_rate):
    if random.random() <= mutation_rate:
        i, j = random.sample(range(len(route)), 2)
        route[i], route[j] = route[j], route[i]
    return route

def genetic_algorithm(cities, distances, population_size=100, generations=100,
mutation_rate=0.01):
    population = generate_initial_population(len(cities), population_size)
    for _ in range(generations):
        population = [mutate(crossover(*random.sample(population, 2)), mutation_rate) for _ in
range(population_size)]
    best_route = min(population, key=lambda route: calculate_total_distance(route, distances))
    return calculate_total_distance(best_route, distances), best_route

cities = ['A', 'B', 'C', 'D']
distances = [
    [0, 10, 15, 20],
    [10, 0, 35, 25],
    [15, 35, 0, 30],
    [20, 25, 30, 0]
]
min_distance, best_route = genetic_algorithm(cities, distances)
print("Shortest distance:", min_distance)
print("Best route:", [cities[i] for i in best_route])
```

OUTPUT



```
Shortest distance: 80
Best route: ['B', 'D', 'C', 'A']
```