Fr. Conceicao Rodrigues College of Engineering Fr. Agnel Ashram, Bandstand, Bandra (W), Mumbai - 400050
Department of Computer Engineering Academic
Term II: 23-24

Class: B.E (Computer), Sem – VI Subject Name: Artificial Intelligence Student

Name: Nimish Ravindra Patil **Roll No: 9565**

Practical No:	**10**
Title:	Simple Prototype for expert system
Date of Performance:	**08/03/2024**
Date of Submission:	**08/04/2024**

Rubrics for Evaluation:

Sr. No	Performance Indicator	Excellent	Good	Below Average	Marks
1	On time Completion & Submission (01)	01 (On Time)	NA	00 (Not on Time)	
2	Logic/Algorithm Complexity analysis (03)	03(Correct)	02(Partial)	01 (Tried)	
3	Coding Standards (03): Comments/indention/Naming conventions Test Cases /Output	03(All used)	02 (Partial)	01 (rarely followed)	
4	Post Lab Assignment (03)	03(done well)	2 (Partially Correct)	1(submitted)	
Total					

Signature of the Teacher:

Source code:

```
import random
```

```
# Genetic Algorithm parameters
POPULATION_SIZE = 50
MUTATION_RATE = 0.01
NUM_GENERATIONS = 1000

# Example city distances
CITY_DISTANCES = [
    [0, 29, 20, 21],
    [29, 0, 15, 18],
    [20, 15, 0, 25],
    [21, 18, 25, 0]
]

def create_initial_population(num_cities):
    population = [] for _ in
    range(POPULATION_SIZE): route =
    list(range(1, num_cities))
    random.shuffle(route)
    population.append(route)
    return population

def calculate_fitness(route):
    total_distance = 0 for i in
    range(len(route) - 1):
        total_distance += CITY_DISTANCES[route[i] - 1][route[i + 1] - 1]
    return total_distance

def crossover(parent1, parent2): offspring = [-1] *
    len(parent1) start_index = random.randint(0, len(parent1)
    - 1) end_index = random.randint(start_index, len(parent1)
    - 1) subset = parent1[start_index:end_index]
    offspring[start_index:end_index] = subset remaining =
    [city for city in parent2 if city not in subset] offspring =
    [city if city == -1 else city for city in offspring] for i in
    range(len(offspring)):
        if offspring[i] == -1:
            offspring[i] = remaining.pop(0)
    return offspring

def mutate(route):
    if random.random() < MUTATION_RATE:
        idx1, idx2 = random.sample(range(len(route)), 2)
        route[idx1], route[idx2] = route[idx2], route[idx1] def
        genetic_algorithm(num_cities):
    population = create_initial_population(num_cities)
    for _ in range(NUM_GENERATIONS):
```

```
        population = sorted(population, key=lambda x: calculate_fitness(x))
        new_population = [] for _ in range(POPULATION_SIZE // 2): parent1, parent2 =
        random.choices(population[:POPULATION_SIZE // 10], k=2) offspring =
        crossover(parent1, parent2) mutate(offspring) new_population.append(offspring)
        population = population[:POPULATION_SIZE // 10] + new_population
    return population[0]

# Example usage: num_cities = 4
optimal_route = genetic_algorithm(num_cities)
print("Optimal Route:", optimal_route)
print("Total Distance:", calculate_fitness(optimal_route))
```

Output: