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: Siddhesh Pradhan **Roll No: 9632**

Practical No:	4
Title:	Travelling Salesman Problem
Date of Performance:	26-02-2024
Date of Submission:	26-02-2024

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:

Experiment No: 4

Title: Travelling salesman problem solving using Genetic Algorithm

Objective: To write a program that solves the traveling Salesman problem in an efficient manner.

Theory:

Given a collection of cities and the cost of travel between each pair of them, the **traveling salesman problem**, or **TSP** for short, is to find the cheapest way of visiting all of the cities and returning to your starting point. In the standard version It study, the travel costs are symmetric in the sense that traveling from city X to city Y costs just as much as traveling from Y to X. Clarity on the problem statement as it may sound simple and deceptive.

Algorithm:

Input

1. Number of cities n
2. Cost of traveling between the cities.
3. $c(i, j)$ $i, j = 1, \ldots, n$.
4. Start with city 1 **Main Steps**

1. /*Initialization */
2. $c \leftarrow 0$
3. Cost $\leftarrow$ 0
4. visits $\leftarrow$ 0
5. $e = 1$ /* pointer of the visited city */
6. For $1 \le r \le n$

   . Do {

      a. Choose pointer j with
      b. minimum = $c(e, j) = \min\{c(e, k)$; visits $(k) = 0$ and $1 \le k \le n$ }
      c. cost $\leftarrow$ cost + minimum - cost
      d. $e = j$

7. $C(r) \leftarrow$ j
8. $C(n) = 1$
9. cost = cost + $c(e, 1)$

Using Genetic Algorithm

Finding a solution to the travelling salesman problem requires setting up a genetic algorithm in a specialized way. For instance, a valid solution would need to represent a route where every location is included at least once and only once. If a route contain a single location more than once, or missed a location out completely it wouldn't be valid and it would be valuable computation time calculating its distance.

Step 1. Choose mutation method to shuffle the route. Note that method should not add routes else invalid solutions will be produced.
Step 2. Select swap mutation for the procedure.
Step3.Select two locations at random to swap their positions.

For example, if swap mutation is applied to the following list, [1,2,3,4,5] it might end up with, [1,2,5,4,3]. Here, positions 3 and 5 were switched creating a new list with exactly the same values, just a different order.

Step 4.Make sure that values are not created and pre-existing values are used.

1	2	3	4	5	6	7	8	9

1	2	8	4	5	6	7	3	9

Step 5. Pick a crossover method which can enforce the same constraint.
Step 6. Select ordered crossover. In this crossover method, select a subset from the first parent, and then add that subset to the offspring.

Step 7. Add any missing values to the offspring from the second parent in order that they are found.
To make this explanation a little clearer consider the following example:

Parents

1	2	3	4	5	6	7	8	9

9	8	7	6	5	4	3	2	1

Offspring

					6	7	8	

9	5	4	3	2	6	7	8	1

Explanation:

Here a subset of the route is taken from the first parent (6,7,8) and added to the offspring's route. Next, the missing route locations are adding in order from the second parent. The first location in the second parent's route is 9 which isn't in the offspring's route so it's added in the first available position. The next position in the parent’s route is 8 which is in the offspring's route so it's skipped. This process continues until the offspring has no remaining empty values. If implemented correctly the end result should be a route which contains all of the positions its parents did with no positions missing or duplicated.

Code:

expt4.py

```
import random
import numpy as np

def create_random_route(num_cities):
    route = list(range(num_cities))
    random.shuffle(route)
    return route

def calculate_total_distance(route, distances):
    total_distance = 0
    for i in range(len(route)):
        total_distance += distances[route[i-1]][route[i]]
    return total_distance

def crossover(parent1, parent2):
    # Order Crossover (OX)
    start, end = sorted(random.sample(range(len(parent1)), 2))
    offspring = [-1] * len(parent1)
    for i in range(start, end + 1):
        offspring[i] = parent1[i]
    for i in range(len(parent2)):
        if parent2[i] not in offspring:
            for j in range(len(offspring)):
                if offspring[j] == -1:
                    offspring[j] = parent2[i]
                    break
    return offspring

def mutate(route):
    # Swap Mutation
    idx1, idx2 = random.sample(range(len(route)), 2)
    route[idx1], route[idx2] = route[idx2], route[idx1]
    return route


def get_distances_from_input():
```

```
    num_cities = int(input("Enter the number of cities: "))
    distances = []
    print("Enter the distances between the cities (separated by spaces):")
    for _ in range(num_cities):
        distances.append(list(map(int, input().split())))
    return np.array(distances)

def genetic_algorithm(distances, population_size=100, num_generations=1000):
    num_cities = len(distances)
    population = [create_random_route(num_cities) for _ in range(population_size)]

    for _ in range(num_generations):
        new_population = []

        # Elitism: Keep the best route from the previous generation
        best_route = min(population, key=lambda x: calculate_total_distance(x, 
distances))
        new_population.append(best_route)

        while len(new_population) < population_size:
            parent1, parent2 = random.choices(population, k=2)
            offspring = crossover(parent1, parent2)
            if random.random() < 0.1:
                offspring = mutate(offspring)
            new_population.append(offspring)

        population = new_population

    best_route = min(population, key=lambda x: calculate_total_distance(x, 
distances))
    best_distance = calculate_total_distance(best_route, distances)
    return best_route, best_distance

if __name__ == "__main__":
    distances = get_distances_from_input()
    best_route, best_distance = genetic_algorithm(distances)
    print(f"Best route: {best_route}")
    print(f"Best distance: {best_distance}")
```

Output:

```
PS C:\Local Disk D\6thSem\AI pracs> python3 expt4.py
Enter the number of cities: 4
Enter the distances between the cities (separated by spaces):
0 10 15 20
10 0 35 25
15 35 0 30
20 25 30 0
Best route: [2, 3, 1, 0]
Best distance: 80
```

Post Lab Assignment:

1. How to overcome combinatorial explosion in TSP?

Ans:- To overcome combinatorial explosion in the Traveling Salesman Problem, heuristic algorithms like Genetic Algorithms or Ant Colony Optimization can be used to efficiently explore the solution space and find near-optimal solutions. Additionally, problem decomposition techniques, such as dividing the cities into clusters and solving smaller subproblems, can reduce the complexity of the overall problem.

2. What is learning from travelling salesperson problem?

Ans:- Learning from the Traveling Salesperson Problem involves using past experiences or solutions to improve future solutions. This can include learning problem-specific insights from previous instances, as well as adapting algorithmic strategies based on the performance of past solutions.