Department of Computer Science and Engineering (Data Science)

Subject: Artificial Intelligence (DJS22DSC502)

AY: 2024-25

Experiment 5

(Solution Space)

Dhruv Shah 60009220132 D1-1

Aim: Implement Genetic Algorithm to solve Travelling Salesman Problem.

Theory:

Genetic algorithms are heuristic search algorithms inspired by the process that supports the evolution of life. The algorithm is designed to replicate the natural selection process to carry generation, i.e. survival of the fittest of beings. Standard genetic algorithms are divided into five phases which are:

1. Creating initial population.
2. Calculating fitness.
3. Selecting the best genes.
4. Crossing over.
5. Mutating to introduce variations.

These algorithms can be implemented to find a solution to the optimization problems of various types. One such problem is the Traveling Salesman Problem. The problem says that a salesman is given a set of cities, he has to find the shortest route to as to visit each city exactly once and return to the starting city. Approach: In the following implementation, cities are taken as genes, string generated using these characters is called a chromosome, while a fitness score which is equal to the path length of all the cities mentioned, is used to target a population.

Fitness Score is defined as the length of the path described by the gene. Lesser the path length fitter is the gene. The fittest of all the genes in the gene pool survive the population test and move to the next iteration. The number of iterations depends upon the value of a cooling variable. The value of the cooling variable keeps on decreasing with each iteration and reaches a threshold after a certain number of iterations.

Algorithm:

1. Initialize the population randomly.

2. Determine the fitness of the chromosome.

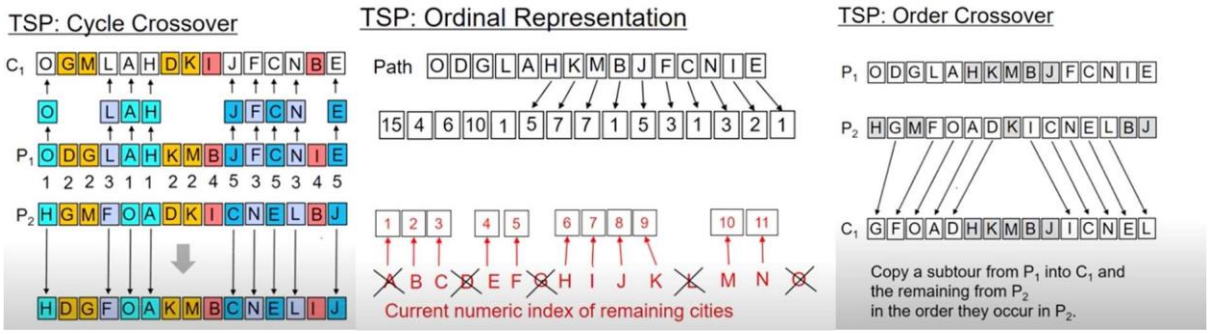
3. Until done repeat:

1. Select parents.

2. Perform crossover and mutation.

3. Calculate the fitness of the new population.

4. Append it to the gene pool.



Lab Assignment to do:

1. Implement Genetic algorithm for 20 cities using Cyclic crossover and find the number of iterations where the algorithm converges.
2. Compare the performance of Genetic algorithm using 5, 10, 20 and 40 cities.

```
import numpy as np
import pandas as pd
import random
no_of_cities = 5
pop_size = 4

def generate_matrix(pop_size, no_of_cities):     dist_matrix =
np.random.randint(1, 100, size=(no_of_cities, no_of_cities))
    np.fill_diagonal(dist_matrix, 0)
return dist_matrix

dist_matrix = generate_matrix(pop_size, no_of_cities)
print("Distance Matrix:\n", dist_matrix)
```

```
Distance Matrix:
 [[ 0 15 20 48 46]
 [62  0 11 37  3]
 [95 67  0 62 33]
 [26 57 24  0 79]
 [65 88 36 65  0]]
```

```
def trip_length(tour, dist_matrix):     return
sum(dist_matrix[tour[i], tour[i + 1]] for i in
range(len(tour) - 1))

def population(pop_size, num_cities):     pop =
[random.sample(range(num_cities), num_cities) for _ in
range(pop_size)]     return pop

def fitness_func(pop, dist_matrix):     return [1 /
trip_length(tour, dist_matrix) for tour in pop]

def probability(fitness):
    total_fitness = sum(fitness)
    return [f / total_fitness for f in fitness]

def cumulative_probability(prob):     cum_prob =
np.cumsum(prob).tolist()  # Get cumulative sum     return
cum_prob

def threshold_selection(pop, prob):
cum_prob = cumulative_probability(prob)
threshold = random.uniform(0, 1)     for i,
p in enumerate(cum_prob):         if
threshold <= p:             return pop[i]

pop = population(pop_size, no_of_cities)
for i, tour in enumerate(pop):
```

```
    print(f"Tour {i + 1}: {tour}")
    print(f"Tour Cost: {trip_length(tour, dist_matrix)}")

fitness = fitness_func(pop, dist_matrix)
prob = probability(fitness)
print("Probabilities:", prob)

selected_tour = threshold_selection(pop, prob)
print("Selected Tour:", selected_tour)
print("Selected Tour Cost:", trip_length(selected_tour, dist_matrix))
```

```
Tour 1: [2, 3, 1, 0, 4]
Tour Cost: 227
Tour 2: [2, 3, 4, 0, 1]
Tour Cost: 221
Tour 3: [4, 3, 2, 1, 0]
Tour Cost: 218
Tour 4: [4, 3, 2, 0, 1]
Tour Cost: 199
Probabilities: [0.2375783782712207, 0.2440284699889914,
0.24738665994296832, 0.27100649179681957]
Selected Tour: [2, 3, 4, 0, 1]
Selected Tour Cost: 221
```

```
def cyclic_crossover(parent1, parent2):
child1 = [-1] * len(parent1)     child2
= [-1] * len(parent2)

    start = 0
index = start

    while child1[start] == -1:
child1[index] = parent1[index]          if
parent2[index] in parent1:              index =
parent1.index(parent2[index])          else:
break

    for i in range(len(parent1)):
if child1[i] == -1:
child1[i] = parent2[i]

    start = 0
index = start

    while child2[start] == -1:
child2[index] = parent2[index]          if
parent1[index] in parent2:              index =
parent2.index(parent1[index])          else:
```

```
            break

    for i in range(len(parent2)):
if child2[i] == -1:
child2[i] = parent1[i]     return
child1, child2

def mutate(tour):     a, b =
random.sample(range(len(tour)), 2)
tour[a], tour[b] = tour[b], tour[a]     return
tour

def select(pop, prob):     return
threshold_selection(pop, prob)

def genetic_algorithm(dist_matrix, pop_size, no_of_cities,
max_iters=1000, mutation_rate=0.1):     pop =
population(pop_size, no_of_cities)     best_tour = None
    best_tour_length = float('inf')
num_iters = 0
    no_improvement_count = 0
    improvement_threshold = 50  # Convergence criterion (if no
improvement in 50 iterations)

    while num_iters < max_iters and no_improvement_count <
improvement_threshold:         fitness = fitness_func(pop,
dist_matrix)         prob = probability(fitness)
new_population = []

        # Generate new population with crossover and mutation
while len(new_population) < pop_size:
            parent1 = select(pop, prob)
parent2 = select(pop, prob)

            child1, child2 = cyclic_crossover(parent1, parent2)

            if random.random() < mutation_rate:
                child1 = mutate(child1)
if random.random() < mutation_rate:
child2 = mutate(child2)

            new_population.extend([child1, child2])

        pop = new_population[:pop_size]  # Ensure the population size
remains the same

        # Check for the best tour in the new population
for tour in pop:
```

```
            length = trip_length(tour, dist_matrix)
if length < best_tour_length:
best_tour = tour                best_tour_length =
length
                no_improvement_count = 0  # Reset improvement count if
there's a new best             else:
no_improvement_count += 1          num_iters += 1

    return best_tour, best_tour_length, num_iters

best_tour, best_tour_length, num_iters =
genetic_algorithm(dist_matrix, pop_size, no_of_cities)

print("Best Tour:", best_tour)
print("Best Tour Length:", best_tour_length)
print("Algorithm Converged after", num_iters, "iterations.")
```

```
Best Tour: [0, 1, 2, 3, 2]
Best Tour Length: 112

Algorithm Converged after 16 iterations.
```

CONCLUSION: We have successfully implemented Genetic Algorithm using Cyclic Crossovers to solve TSP.