Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

Subject: Artificial Intelligence (DJ19DSC502)

AY: 2023-24

Jhanvi Parekh

60009210033

D11

Experiment 5

(Solution Space)

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.

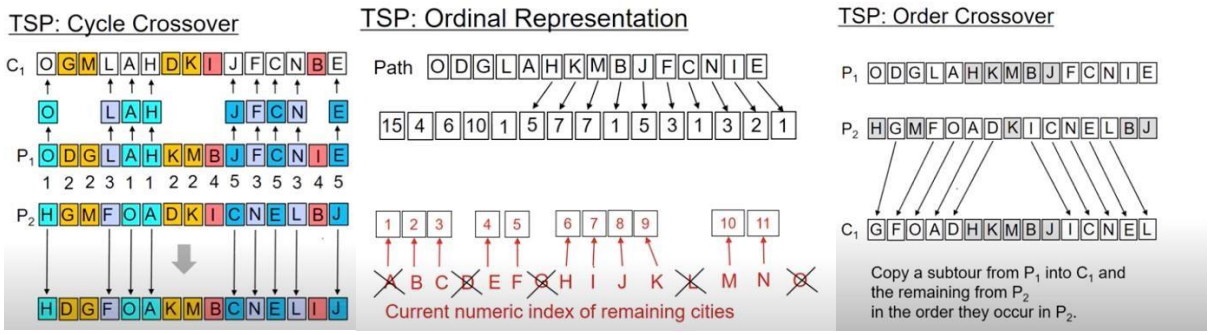
3. Until done repeat:

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:

LINK : https://colab.research.google.com/drive/1D-zwZfoXrD-YK1yzaVdjyj0VlZaAotBY?usp=sharing

```
import itertools
import random

cities = ["A", "B", "C", "D", "E"]

permutations = list(itertools.permutations(cities, 5))
combinations = list(itertools.combinations(cities, 5))
unique_solutions = set(permutations + combinations)
all_solutions = list(unique_solutions)
random.shuffle(all_solutions)


for i, solution in enumerate(all_solutions, start=1):
    print(f"Solution {i}: {', '.join(solution)}")
selected_solutions = random.sample(all_solutions, 5)

print("\nRandomly Selected Solutions:")
for i, solution in enumerate(selected_solutions, start=1):
    print(f"Solution {i}: {', '.join(solution)}")
```

```
Solution 1: E, A, B, D, C
Solution 2: D, C, E, B, A
Solution 3: A, B, D, E, C
Solution 4: B, A, E, D, C
Solution 5: D, A, E, B, C
Solution 6: E, B, C, D, A
Solution 7: E, B, D, C, A
Solution 8: D, E, A, C, B
Solution 9: D, C, A, E, B
Solution 10: C, B, D, E, A
Solution 11: A, D, C, B, E
Solution 12: C, D, A, B, E
Solution 13: C, D, A, E, B
Solution 14: A, D, B, E, C
Solution 15: B, E, D, C, A
Solution 16: D, A, B, E, C
Solution 17: B, A, C, E, D
Solution 18: B, C, E, A, D
Solution 19: C, B, A, E, D
Solution 20: B, D, C, E, A
Solution 21: C, E, B, D, A
Solution 22: A, B, D, C, E
Solution 23: A, E, B, C, D
Solution 24: A, D, E, B, C
Solution 25: B, E, A, C, D
Solution 26: E, B, A, C, D
Solution 27: A, C, D, B, E
Solution 28: B, E, C, D, A
Solution 29: A, B, C, D, E
Solution 30: D, A, E, C, B
Solution 31: E, B, A, D, C
Solution 32: C, E, B, A, D
Solution 33: E, A, D, C, B
Solution 34: B, C, E, D, A
Solution 35: D, B, E, A, C
Solution 36: E, D, C, A, B
Solution 37: D, B, C, A, E
Solution 38: E, D, B, C, A
Solution 39: A, E, C, B, D
Solution 40: E, C, B, D, A
Solution 41: B, A, D, E, C
Solution 42: A, D, C, E, B
Solution 43: E, A, C, D, B
Solution 44: B, E, D, A, C
Solution 45: D, A, B, C, E
Solution 46: E, D, A, B, C
Solution 47: A, B, C, E, D
Solution 48: C, D, B, E, A
Solution 49: D, B, A, E, C
Solution 50: A, E, D, B, C
Solution 51: D, E, C, B, A
Solution 52: E, A, B, C, D
Solution 53: E, D, B, A, C
Solution 54: A, D, B, C, E
Solution 55: E, C, A, D, B
Solution 56: C, A, B, E, D
Solution 57: C, E, D, B, A
Solution 58: C, D, E, B, A
```

```
import itertools
```

```
import random


distance_matrix = {}
for city1 in cities:
    for city2 in cities:
        if city1 != city2:
            distance = int(input(f"Enter distance between {city1} and {city2}: "))
            distance_matrix[(city1, city2)] = distance

"""
distance_matrix = {
  ("A", "B"): 10,
  ("A", "C"): 15,
  ("A", "D"): 20,
  ("A", "E"): 25,
  ("B", "A"): 10,
  ("B", "C"): 35,
  ("B", "D"): 30,
  ("B", "E"): 10,
  ("C", "A"): 15,
  ("C", "B"): 35,
  ("C", "D"): 10,
  ("C", "E"): 20,
  ("D", "A"): 20,
  ("D", "B"): 30,
  ("D", "C"): 10,
  ("D", "E"): 15,
  ("E", "A"): 25,
  ("E", "B"): 10,
  ("E", "C"): 20,
  ("E", "D"): 15,
}
"""
def calculate_distance(solution):
    total_distance = 0
    for i in range(len(solution) - 1):
        city1, city2 = solution[i], solution[i + 1]
        total_distance += distance_matrix[(city1, city2)]
    return total_distance

print("\nRandomly Selected Solutions:")
for i, solution in enumerate(selected_solutions, start=1):
    distance = calculate_distance(solution)
    print(f"Solution {i}: {', '.join(solution)} - Total Distance: {distance}")

     Enter distance between A and B: 10
     Enter distance between A and C: 15
     Enter distance between A and D: 20
     Enter distance between A and E: 25
     Enter distance between B and A: 10
     Enter distance between B and C: 35
     Enter distance between B and D: 30
     Enter distance between B and E: 10
     Enter distance between C and A: 15
     Enter distance between C and B: 35
     Enter distance between C and D: 10
     Enter distance between C and E: 20
     Enter distance between D and A: 20
     Enter distance between D and B: 30
     Enter distance between D and C: 10
     Enter distance between D and E: 15
     Enter distance between E and A: 25
     Enter distance between E and B: 10
     Enter distance between E and C: 20
     Enter distance between E and D: 15

     Randomly Selected Solutions:
     Solution 1: B, D, C, E, A - Total Distance: 85
     Solution 2: E, A, C, D, B - Total Distance: 80
     Solution 3: D, C, B, A, E - Total Distance: 80
     Solution 4: B, D, E, C, A - Total Distance: 80
     Solution 5: A, D, B, C, E - Total Distance: 105


def mutate_solution(solution):
    mutated_solution = list(solution)
    index1, index2 = random.sample(range(len(mutated_solution)), 2)
    mutated_solution[index1], mutated_solution[index2] = mutated_solution[index2], mutated_solution[index1]
    return tuple(mutated_solution)
```

```
mutated_solutions = [mutate_solution(solution) for solution in selected_solutions]

print("\nMutated Solutions:")
for i, solution in enumerate(mutated_solutions, start=1):
    distance = calculate_distance(solution)
    print(f"Solution {i}: {', '.join(solution)} - Total Distance: {distance}")

print("\nSolutions with Total Distance <= 80:")
for i, solution in enumerate(mutated_solutions, start=1):
    distance = calculate_distance(solution)
    if distance <= 80:
        print(f"Solution {i}: {', '.join(solution)} - Total Distance: {distance}")
```

```
Mutated Solutions:
Solution 1: C, B, E, A, D - Total Distance: 90
Solution 2: E, B, D, C, A - Total Distance: 65
Solution 3: C, D, B, A, E - Total Distance: 75
Solution 4: E, A, D, B, C - Total Distance: 110
Solution 5: E, A, C, B, D - Total Distance: 105

Solutions with Total Distance <= 80:
Solution 2: E, B, D, C, A - Total Distance: 65
Solution 3: C, D, B, A, E - Total Distance: 75
```

```
def crossover(parent1, parent2):
    point1, point2 = sorted(random.sample(range(len(parent1)), 2))
    child = parent1[point1:point2]
    for city in parent2:
        if city not in child:
            child += (city,)

    return child

parent1_index, parent2_index = random.sample(range(len(mutated_solutions)), 2)
parent1 = mutated_solutions[parent1_index]
parent2 = mutated_solutions[parent2_index]

child_solution = crossover(parent1, parent2)

print(f"Parent 1: {', '.join(parent1)}")
print(f"Parent 2: {', '.join(parent2)}")
print(f"Child: {', '.join(child_solution)}")
```

```
Parent 1: E, A, D, B, C
Parent 2: E, A, C, B, D
Child: A, D, B, E, C
```