Page No.
Date

Devanshu Surana
PC-23, 1032210755
Panel C, Batch C1

AIES lab Assignment 5

Aim: Implement Hill Climbing algorithm for TSP

Objective: Write C|C++|Java|Python to solve hill climb algorithm for travelling salesman problem.

Theory:
1. Local Search Algorithm:
   1. Used for optimization problems
   2. Focuses on exploring solutions within a localized region.
   3. Continously iterating by evaluating and selecting neighbourhood solutions.
   4. Stops when no better solution can be found in local vicinity.

2. Hill Climbing Algorithm:
   1. A type of local search algorithm
   2. Begins with an initial soln.
   3. Repeatedly makes small adjustments to reach better soln.
   4. Halts when it reaches a peak where no single step improvement is possible.

Input: n x n matrix of distance for TSP.
Output: An optimal distance betn two cities.

Page No.
Date

Algo: Hill Climbing Algorithm.

FAQ's

1. Explain Hill Climbing Algorithm in detail with example.

→ Hill climbing is a local search algorithm used for optimization. It starts from an initial stage (solution) and iteratively moves to neighbouring solutions with better objective values until it reaches local maximusm.

Ex)

2 8 3 (start)
1 6 4 f(n) = -4
7 □ 5

1 2 3 (goal)
8 □ 4
7 6 5

-5 ↓ -5

2 8 3
1 □ 4
7 6 5

-3 ↓ ~~-4~~

-2 ↑

1 2 3
□ 8 4 f(n) = -1
7 6 5

↑

2 □ 3
f(n) = -3 1 8 4
7 6 5
-4

→

□ 2 3
1 8 4 f(n) = -2
7 6 5

2. Explain limitations of hill climbing and solutions to it.

→ 1. Local Maxima/Minima: Hill climbing can get stuck in local maximus and fail to reach global maximum. Solution include random restarts; simulated annealing and genetic algorithms to explore beyond local maxima.

2) Plateous and Ridges: On plateaus or step, ridges of the search space, hill climbing may progress slowly. Using stages can solve this issue. (Tabu search and simulated annealing)

3) Choice of Initial stage: Performance can vary based on the initial solution. Using multiple initial states can solve this issue.

4) Premature Convergence: Hill climbing may converge too quickly, missing better solution. Diversification strategies and adaptive step size can mitigate this.

3. Solve N-Queen problem using local search algorithm.

→

```
Q - - -      (start)
- Q - -      f(n) = -2
- Q - -
- - Q -
   ↓
Q - - -
- - Q -      f(n) = -2
- Q - -
- - Q -
   ↓
Q - - -
- - - Q      f(n) = -1
- Q - Q      →
- - Q -
```

4 Queen Problem solved using Hill Climbing Algorithm.

```
Q - - -
- - - Q      (goal)
Q - - -      f(n) = 0
- - Q -
```

29/11/23

```python
import random

# Function to create a random solution generator
def randomSolution(num_cities):
    cities = list(range(num_cities))
    solution = random.sample(cities, num_cities)
    return solution

# Function for calculating the length of a route
def routeLength(tsp, solution):
    route_length = 0
    num_cities = len(tsp)
    for i in range(num_cities):
        route_length += tsp[solution[i - 1]][solution[i]]
    return route_length

# Function for generating all neighbors of a solution
def getNeighbours(solution):
    neighbours = []
    num_cities = len(solution)
    for i in range(num_cities):
        for j in range(i + 1, num_cities):
            neighbour = solution[:]
            neighbour[i] = solution[j]
            neighbour[j] = solution[i]
            neighbours.append(neighbour)
    return neighbours

# Function for finding the best neighbor
def getBestNeighbour(tsp, neighbours):
    best_route_length = routeLength(tsp, neighbours[0])
    best_neighbour = neighbours[0]
    for neighbour in neighbours:
        current_route_length = routeLength(tsp, neighbour)
        if current_route_length < best_route_length:
            best_route_length = current_route_length
            best_neighbour = neighbour
    return best_neighbour, best_route_length

# Hill climbing algorithm
def hillClimbing(tsp, num_cities):
    current_solution = randomSolution(num_cities)
    current_route_length = routeLength(tsp, current_solution)
    neighbours = getNeighbours(current_solution)
    best_neighbour, best_neighbour_route_length =
getBestNeighbour(tsp, neighbours)

    while best_neighbour_route_length < current_route_length:
        current_solution = best_neighbour
        current_route_length = best_neighbour_route_length
```

```
        neighbours = getNeighbours(current_solution)
        best_neighbour, best_neighbour_route_length =
getBestNeighbour(tsp, neighbours)

    return current_solution, current_route_length

def main():
    num_cities = int(input("Enter the number of cities: "))
    tsp = []

    for i in range(num_cities):
        row = list(map(int, input(f"Enter the distances from city
{i+1} to all cities separated by spaces: ").split()))
        tsp.append(row)

    solution, route_length = hillClimbing(tsp, num_cities)

    print("Optimal Route:", solution)
    print("Optimal Route Length:", route_length)

if __name__ == "__main__":
    main()
```

```
Enter the number of cities: 5
Enter the distances from city 1 to all cities separated by spaces: 10
12 13 19 9
Enter the distances from city 2 to all cities separated by spaces: 12
13 10 7 9
Enter the distances from city 3 to all cities separated by spaces: 13
14 12 10 8
Enter the distances from city 4 to all cities separated by spaces: 12
13 13 10 7
Enter the distances from city 5 to all cities separated by spaces: 12
13 10 8 9
Optimal Route: [0, 1, 3, 4, 2]
Optimal Route Length: 49
```