Topic :

Page No. :
Date. : / /

Assignment - 06 [AIES]

* Tittle :
Implement a local search algorithm for eg : n-queens, chess, TSP

* Aim :
Implement hill climbing algorithm for TSP

* Objective :
Write a program in C/C++/Python to solve the hill climbing algorithm for TSP.

* Theory :

1.) Local Search Algorithm
local search algorithm are optimization techniques used to find solutions by exploring the solution space incrementally. These algorithms start with an initial solution and iteratively make small changes to it

Key Characteristics :

- Focuses on a single current solution.
- Moves to a neighbouring solution if it improves the current one.
- Finds a good solution in a ~~rese~~ reasonable time.

For Educational Use Only

Topic : _____ Page No. : ____ Date. : / /

2) Hill Climbing Algorithm

Hill climbing is a specific type of local search algorithm that continuously moves towards higher (or lower) values in the search space.
It is an iterative algorithm that starts with an arbitary solution and makes incremental changes to improve it.

Key Characteristics :
- Greedy Approach
- Simple and easy to implement

Types of Hill Climbing :
- Single hill climbing
- Steepest hill climbing
- Stohastic hill climbing

* Input :
n * n metrix of distance of TSP.

* Output :
An optimal distance between 2 cities

* Algorithm :
Hill Climbing Algorithm

For Educational Use Only

Topic :

Page No. :
Date. : / /

* Platform :
linux

* FAQ's :

Q.1. Explain hill climbing with example.

→ Hill climbing is a specific type of local search algorithm that continuously moves towards higher (or lower) values in the search space.
It is a heuristic search technique used for mathematical ~~operation~~ optimization problems.
Steps of hill climbing algorithm :

(i) Initialisation
(ii) Evaluation
(iii) Generate neighbour
(iv) Select best neighbour
(v) Move to ~~next~~ best neighbour
(vi) Check for termination

Example : $f(x) = -x^2 + 4x$

(i) Initialize $x = 0$
(ii) Evaluate :
$f(x=0) = 0$

For Educational Use Only

Topic : _____ Page No. : ____ Date. : / /

(iii) Generate neighbours
$f(x=-1) = 3$
$f(x=1) = 5$

(iv) Thus best neighbour ($x = -1$)

(v) Move to $x=1$

(vi) Repeat steps (iii) - (v)

(vii) Here, the algorithm terminates at $x=2$. with max function value $f(2)=4$

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

→ limitations :
- Hill climbing can get stuck in local optima.
- Hill climbing can get stuck on plateaus, flat areas of the search space where neighbouring points have same value
- Hill climbing does not consider the possibility of backtracking.
- The quality of final sol^n depends heavily on starting point.

Solutions to overcome limitations :
- Run the algorithm multiple times from different random starting points.
- Simulated Annealing

For Educational Use Only

Topic :

Page No. :
Date. : / /

- Genetic Algorithms : These use principles of natural selection and genetics to search for optimal solutions.
- Beam Search and Tabu Search.

Q.3. Solve n queens problem using local search algorithm.

→ The n queens problem is a classic combinatorial problem where the goal is to place N queens on an N×N chessboard such that no two queens attack each other.

Steps to solve N-queens problem :

(i) Initialisation : Start with a random placement of n queens on an N×N Chessboard.

(ii) Iterate :
For a given no. of conflicts :
- Select a queen in this conflict
- Move the selected queen to a position that minimizes the conflict

(iii.) Termination :
If no conflicts are present, the solution is found.

For Educational Use Only

Topic : _____ Page No. : ____ Date. : / /

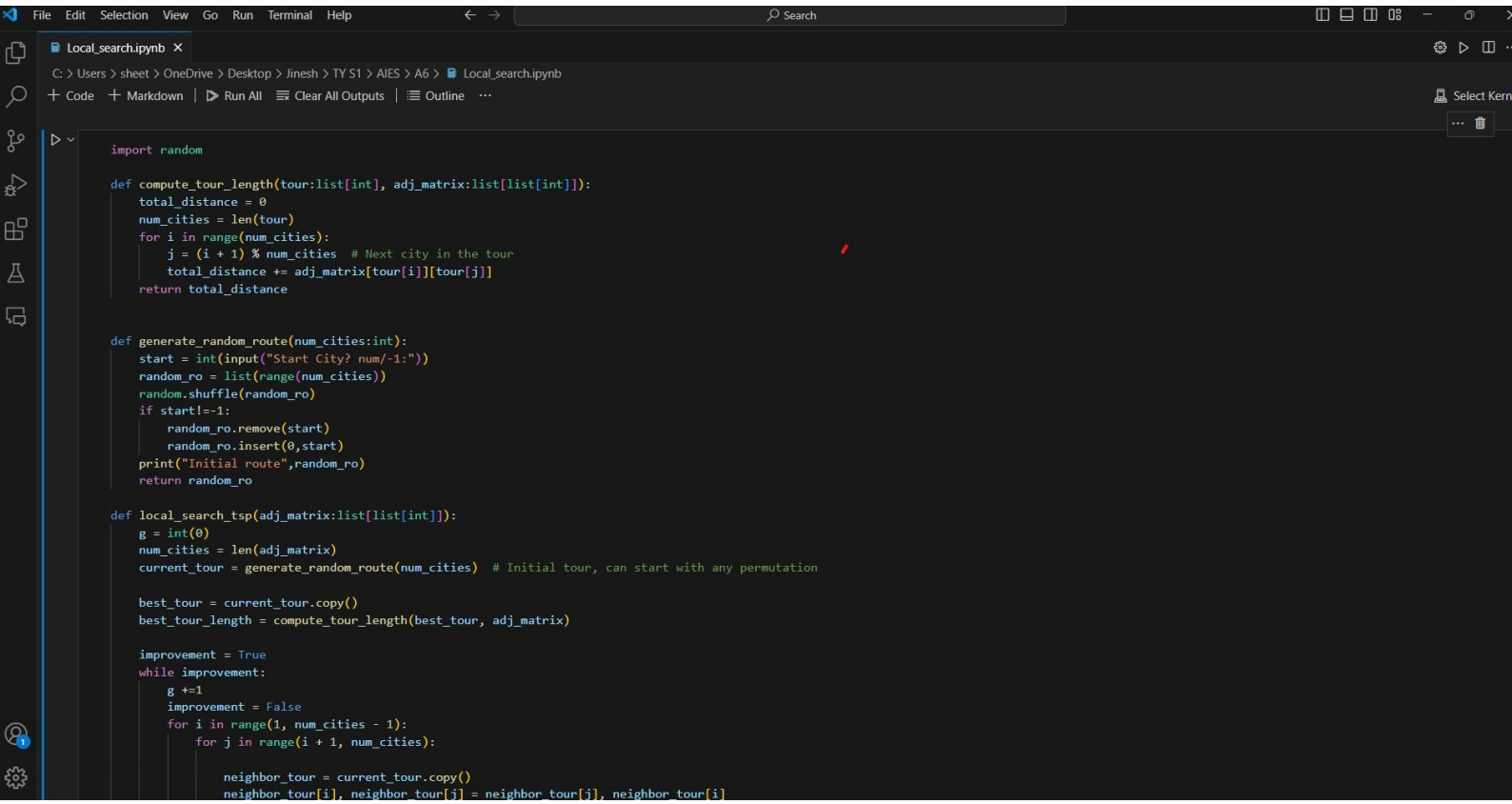
* Algorithm :-

```
def hill_climbing (problem):
    current_solution = problem.initial_solution
                                    ()

    while True:
        neighbours = problem.generate_
                     neighbours (current_
                                 solution)

        next_solution = None
        for neighboure in neighbours:
            if problem.evaluate (neighbours) >
               problem.evaluate (current_soln):
                next_solution = neighbour
                break :
        if next_solution is None:
            return current_solution
        current_solution = next_solution
```

* Conclusion :-
We learned hill climbing algorithm, local search algorithm and TSP.

30/7/24

For Educational Use Only


File Edit Selection View Go Run Terminal Help
Search
Local_search.ipynb
C: > Users > sheet > OneDrive > Desktop > Jinesh > TY S1 > AIES > A6 > Local_search.ipynb
+ Code + Markdown | Run All Clear All Outputs | Outline ...
Select Kern
import random
def compute_tour_length(tour:list[int], adj_matrix:list[list[int]]):
total_distance = 0
num_cities = len(tour)
for i in range(num_cities):
j = (i + 1) % num_cities # Next city in the tour
total_distance += adj_matrix[tour[i]][tour[j]]
return total_distance
def generate_random_route(num_cities:int):
start = int(input("Start City? num/-1:"))
random_ro = list(range(num_cities))
random.shuffle(random_ro)
if start!=-1:
random_ro.remove(start)
random_ro.insert(0,start)
print("Initial route",random_ro)
return random_ro
def local_search_tsp(adj_matrix:list[list[int]]):
g = int(0)
num_cities = len(adj_matrix)
current_tour = generate_random_route(num_cities) # Initial tour, can start with any permutation
best_tour = current_tour.copy()
best_tour_length = compute_tour_length(best_tour, adj_matrix)
improvement = True
while improvement:
g +=1
improvement = False
for i in range(1, num_cities - 1):
for j in range(i + 1, num_cities):
neighbor_tour = current_tour.copy()
neighbor_tour[i], neighbor_tour[j] = neighbor_tour[j], neighbor_tour[i]

```
                neighbor_tour = current_tour.copy()
                neighbor_tour[i], neighbor_tour[j] = neighbor_tour[j], neighbor_tour[i]
                neighbor_tour_length = compute_tour_length(neighbor_tour, adj_matrix)


                if neighbor_tour_length < best_tour_length:
                    current_tour = neighbor_tour.copy()
                    best_tour = current_tour.copy()
                    best_tour_length = neighbor_tour_length
                    print("Selected improved neighbour",current_tour)
                    improvement = True
                    break

            if improvement:
                break

    return best_tour, best_tour_length,g

adjacency_matrix = [
    [0, 400, 500, 300],
    [400, 0, 300, 500],
    [500, 300, 0, 400],
    [300, 500, 400, 0]
]

best_tour, best_tour_length ,total_steps= local_search_tsp(adjacency_matrix)
print("Best Tour:", best_tour)
print("Tour Length:", best_tour_length)
print("Total Steps:", total_steps)
```

[3] Python

```
Initial route [2, 1, 3, 0]
Selected improved neighbour [2, 1, 0, 3]
Best Tour: [2, 1, 0, 3]
Tour Length: 1400
Total Steps: 2
```

Ln 1, Col 1 Spaces: 4 UTF-8 CRLF Plain Text