Class and Branch: TE AI & DS
Student Roll no.: A07
Student Name: Niharika Seth

Date: 04/10/2024

EXPERIMENT NO.: 07

AIM: To implement Hill Climbing Search algorithm in AI with python code

OBJECTIVES: To find the peak value or best solution to the problem.

THEORY:

Hill climbing is a widely used optimization algorithm in Artificial Intelligence (AI) that helps find the best possible solution to a given problem. As part of the local search algorithms family, it is often applied to optimization problems where the goal is to identify the optimal solution from a set of potential candidates.

Understanding Hill Climbing in AI
Hill Climbing is a heuristic search algorithm used primarily for mathematical optimization problems in artificial intelligence (AI). It is a form of local search, which means it focuses on finding the optimal solution by making incremental changes to an existing solution and then evaluating whether the new solution is better than the current one. The process is analogous to climbing a hill where you continually seek to improve your position until you reach the top, or local maximum, from where no further improvement can be made.

Hill climbing is a fundamental concept in AI because of its simplicity, efficiency, and effectiveness in certain scenarios, especially when dealing with optimization problems or finding solutions in large search spaces.

Features of Hill Climbing:

1. Variant of Generating and Testing Algorithm: Hill Climbing is a specific variant of the generating and testing algorithms.
   The process involves:

- Generating possible solutions: The algorithm creates potential solutions within the search space.
- Testing solutions: Each generated solution is evaluated to determine if it meets the desired criteria.
- Iteration: If a satisfactory solution is found, the algorithm terminates; otherwise, it returns to the generation step.
- This iterative feedback mechanism allows Hill Climbing to refine its search by using information from previous evaluations to inform future moves in the search space.

2. Greedy Approach: The Hill Climbing algorithm employs a greedy approach, meaning that at each step, it moves in the direction that optimizes the objective function. This strategy aims to find the optimal solution efficiently by making the best immediate choice without considering the overall problem context.

Problem Faced with Hill Climbing Algorithm

1. Local Maximum Problem: A local maximum occurs when all neighboring states have worse values than the current state. Since Hill Climbing uses a greedy approach, it will not move to a worse state, causing the algorithm to terminate even though a better solution may exist further along.

   How to Overcome Local Maximum?

- Backtracking Techniques: One effective way to overcome the local maximum problem is to use backtracking. By maintaining a list of visited states, the algorithm can backtrack to a previous configuration and explore new paths if it reaches an undesirable state.

2. Plateau Problem: A plateau is a flat region in the search space where all neighboring states have the same value. This makes it difficult for the algorithm to choose the best direction to move forward.

   How to Overcome Plateau?

- Random Jumps: To escape a plateau, the algorithm can make a significant jump to a random state far from the current position. This increases the likelihood of landing in a non-plateau region where progress can be made.

3. Ridge Problem: A ridge is a region where movement in all possible directions seems to lead downward, resembling a peak. As a result, the Hill Climbing algorithm may stop prematurely, believing it has reached the optimal solution when, in fact, better solutions exist.

   How to Overcome Ridge?

- Multi-Directional Search: To overcome a ridge, the algorithm can apply two or more rules before testing a solution. This approach allows the algorithm to move in multiple directions simultaneously, increasing the chance of finding a better path.

How does the Hill Climbing Algorithm work?

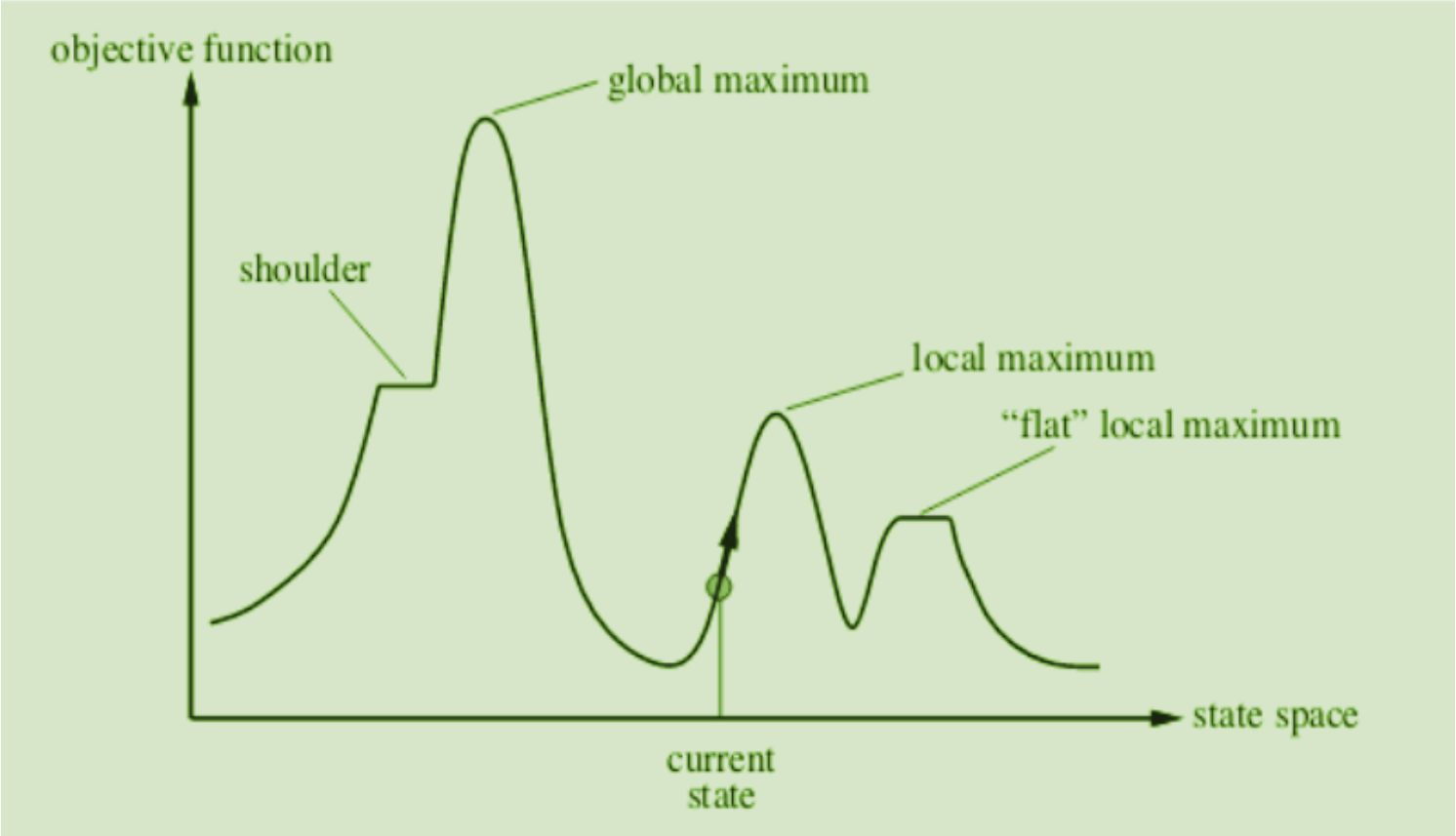


Fig: State Space Diagram In Hill Climbing

The Hill Climbing Algorithm is a heuristic search algorithm used for mathematical optimization problems. It belongs to the class of local search algorithms, which means it explores the solution space by starting from an arbitrary solution and iteratively making small changes to improve it.

Class and Branch: TE AI & DS
Student Roll no.: A07
Student Name: Niharika Seth

Date: 04/10/2024

How it Works:

Step 1- Initial State: The algorithm starts from a randomly selected solution (also called the *current state*).

Step 2- Evaluation: It evaluates the neighboring states of the current solution. A neighboring state is one that can be reached by making a small change to the current solution.

Step 3- Move to a Better State: Among the neighboring states, the one with the best improvement is chosen. If a neighboring state has a better score (based on the evaluation function), the algorithm moves to that state.

Step 4- Repeat: Steps 2 and 3 are repeated until no better neighboring state is found. This happens when the algorithm reaches a local maximum, plateau, or global maximum.

Step 5- Termination: The algorithm terminates when no better neighboring solution is found, indicating that it is at a peak (local or global).

Key Concepts:

- Objective Function: The function to be optimized (maximized or minimized). This could be anything from minimizing distance in a pathfinding problem to maximizing profit in a business model.
- Local Maximum: A point in the solution space where no neighboring solutions have a better score, but it's not the overall best solution (global maximum).
- Global Maximum: The highest point in the entire solution space, which represents the best possible solution.
- Plateau: A flat region in the solution space where neighboring states have the same score, making it hard to decide the direction of search.

Example: Consider the matrix for our Hill Climbing search algorithm as follows:

16	1	23	20	14	28	23
12	17	6	22	29	2	1
20	10	20	16	21	21	9
10	22	5	11	5	22	2
13	29	26	7	11	26	18

Step 1 : Take the start state as (2 , 5).
The number at (2,5) is 21.

Step 2: Now, we search for a greater number in the 3 x 3 matrix that surrounds the number 21.

29	2	1
21	21	9
5	22	2

We find the largest number in this sub-matrix is 29 which is present at (1,4) of the original matrix.

Class and Branch: TE AI & DS
Student Roll no.: A07
Student Name: Niharika Seth

Date: 04/10/2024

Step 3: Now, consider the submatrix around 29 and find the largest number.

20	14	28
22	29	2
16	21	21

Since, there is no larger number than 29 in this sub-matrix.
Therefore, our end state is 29 at location (1 ,4) of the original matrix.

PROGRAM:

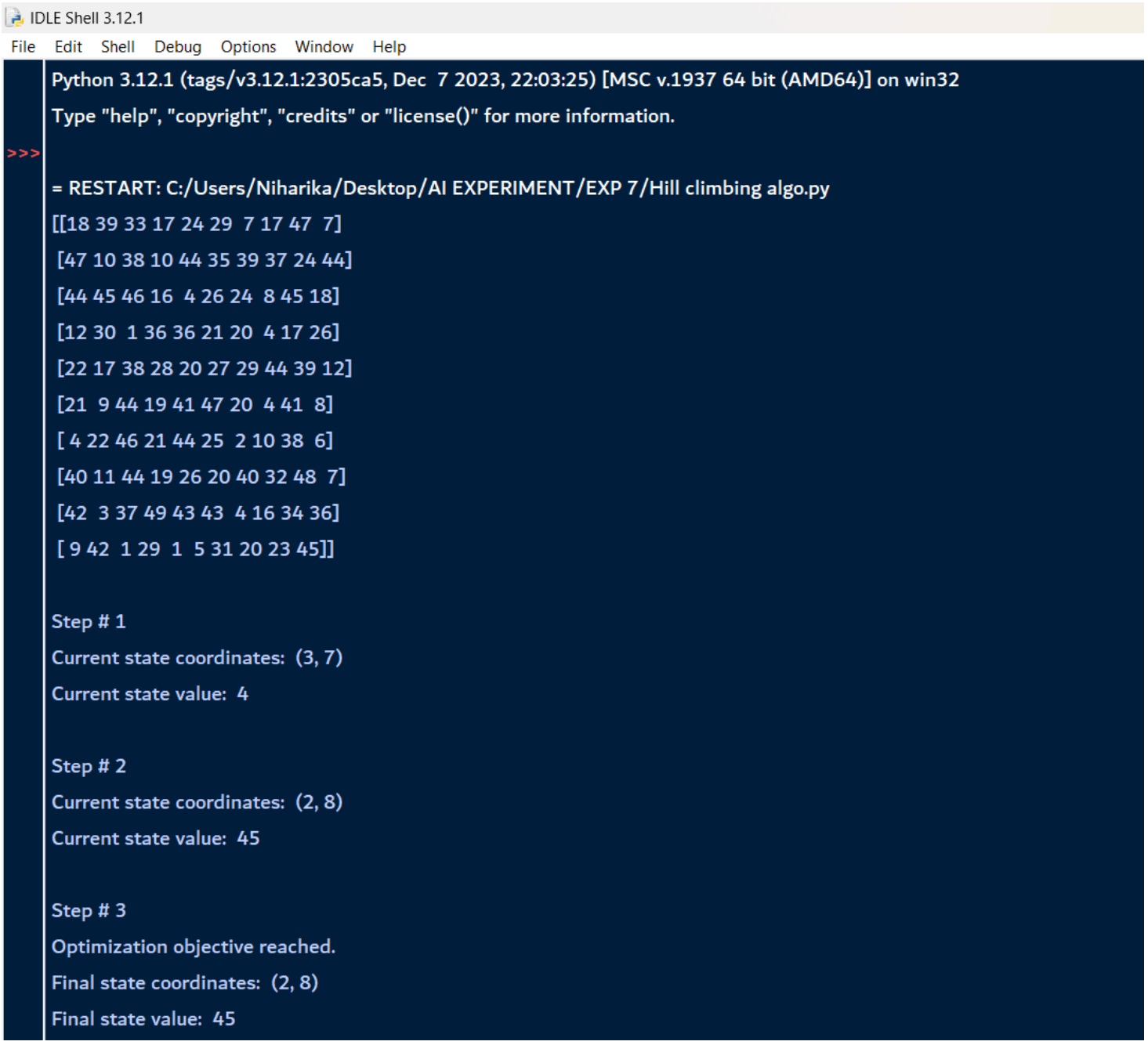
```
# Hill Climbing Algorithm
import numpy as np
def find_neighbours(state, landscape):
    neighbours = []
    dim = landscape.shape
    # top neighbour
    if state[0] != 0:
        neighbours.append((state[0] - 1, state[1]))
    # bottom neighbour
    if state[0] != dim[0] - 1:
        neighbours.append((state[0] + 1, state[1]))
    # left neighbour
    if state[1] != 0:
        neighbours.append((state[0], state[1] - 1))
    # right neighbour
    if state[1] != dim[1] - 1:
        neighbours.append((state[0], state[1] + 1))
    # top left
    if state[0] != 0 and state[1] != 0:
        neighbours.append((state[0] - 1, state[1] - 1))
    # top right
    if state[0] != 0 and state[1] != dim[1] - 1:
        neighbours.append((state[0] - 1, state[1] + 1))
    # bottom left
    if state[0] != dim[0] - 1 and state[1] != 0:
        neighbours.append((state[0] + 1, state[1] - 1))
```

```
    # bottom right
    if state[0] != dim[0] - 1 and state[1] != dim[1] - 1:
        neighbours.append((state[0] + 1, state[1] + 1))
    return neighbours
# Current optimization objective: local/global maximum
def hill_climb(curr_state, landscape):
    neighbours = find_neighbours(curr_state, landscape)
    bool
    ascended = False
    next_state = curr_state
    for neighbour in neighbours: #Find the neighbour with the greatest value
        if landscape[neighbour[0]][neighbour[1]] > landscape[next_state[0]][next_state[1]]:
            next_state = neighbour
            ascended = True
    return ascended, next_state
def __main__():
    landscape = np.random.randint(1, high=50, size=(10, 10))
    print(landscape)
    start_state = (3, 7)  # matrix index coordinates
    current_state = start_state
    count = 1
    ascending = True
    while ascending:
        print("\nStep #", count)
        print("Current state coordinates: ", current_state)
        print("Current state value: ", landscape[current_state[0]][current_state[1]])
        count += 1
        ascending, current_state = hill_climb(current_state, landscape)
    print("\nStep #", count)
    print("Optimization objective reached.")
    print("Final state coordinates: ", current_state)
    print("Final state value: ", landscape[current_state[0]][current_state[1]])
__main__()
```

Class and Branch: TE AI & DS
Student Roll no.: A07
Student Name: Niharika Seth

Date: 04/10/2024

OUTPUT:



CONCLUSION:

The Hill Climbing Search Algorithm effectively demonstrates a heuristic approach to solving optimization problems like the N-Queens problem. Its simplicity allows for easy implementation and efficient solutions for small-scale problems by incrementally evaluating neighboring states.
However, the algorithm's greedy nature can lead to local maxima or plateaus, limiting its effectiveness. While it finds solutions quickly, modifications like random restarts or alternative methods such as Simulated Annealing can enhance its performance.

****