Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

AY: 2025-26

Class:	**TE**	**Semester:**	**V**
Course Code:	**CSL502**	**Course Name:**	**Artificial Intelligence Lab**

Name of Student:	**Shravani Sandeep Raut**
Roll No. :	**51**
Experiment No.:	**5**
Title of the Experiment:	**Implement Hill Climbing Search for problem solving**
Date of Performance:	**05/08/2025**
Date of Submission:	**12/08/2025**

Evaluation

Performance Indicator	Max. Marks	Marks Obtained
Performance	5	
Understanding	5	
Journal work and timely submission	10	
Total	20	

Performance Indicator	Exceed Expectations (EE)	Meet Expectations (ME)	Below Expectations (BE)
Performance	4-5	2-3	1
Understanding	4-5	2-3	1
Journal work and timely submission	8-10	5-8	1-4

Checked by

Name of Faculty : Mrs. Rujuta Vartak

Signature :

Date:

Aim: Implement Hill Climbing Search for problem solving.

Objectives:

Understand the concept of Hill Climbing as a local search algorithm**.**

Theory:

Hill Climbing is a heuristic local search algorithm used for mathematical optimization and artificial intelligence problems. It iteratively moves to neighboring states that improve the evaluation function (heuristic) value and halts when no further improvements can be found.

It works on the principle of greedy ascent, always choosing the best local move based on a heuristic function. However, it can get stuck in:

- Local maxima: A state better than neighbors but not optimal.
- Plateaus: A flat region where all neighboring states have the same value.
- Ridges: A narrow path to the optimal state that requires indirect moves.

Variants of Hill Climbing:

1. Simple Hill Climbing: Considers one neighbor at a time.
2. Steepest-Ascent Hill Climbing: Evaluates all neighbors and chooses the best.
3. Stochastic Hill Climbing: Randomly selects one of the better neighbors.
4. Random Restart Hill Climbing: Runs the algorithm multiple times with different start states.

Pseudocode: Hill Climbing Search

```
HILL-CLIMBING(problem):
    current ← INITIAL-STATE(problem)
    loop do:
        neighbors ← EXPAND(current)
        if neighbors is empty:
            return current   // No more moves possible
        next ← a neighbor with the highest VALUE among neighbors
```

```
if VALUE(next) ≤ VALUE(current):
    return current   // No improvement, return current state
    current ← next       // Move to the better neighbor
```

Explanation of Terms:

- INITIAL-STATE(problem): Starting state of the problem.
- EXPAND(current): Returns all possible neighboring states from the current state.
- VALUE(state): Heuristic function to evaluate the quality of a state.
- The loop continues as long as a better neighbor is found.
- It **stops** when no neighbor is better than the current state (i.e., local maximum or plateau).

Advantages:

- Space-efficient (uses constant memory).
- Fast for simple problems.

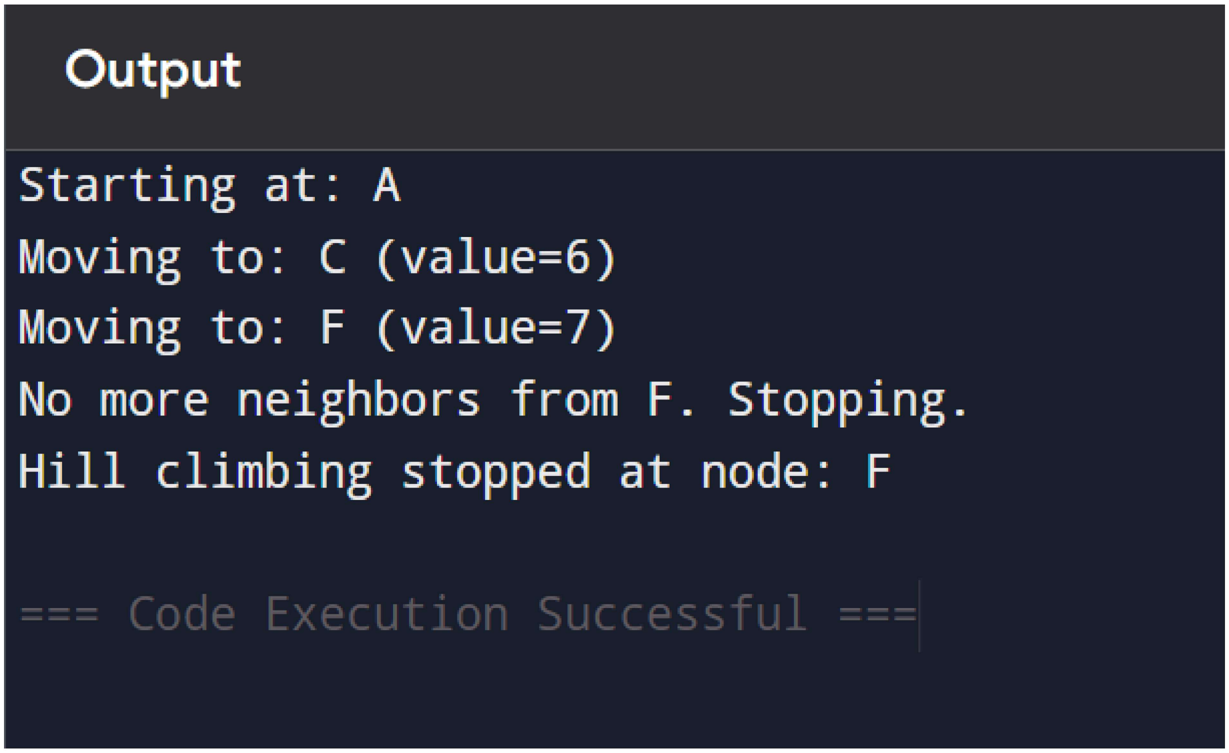
Disadvantages:

- Can get stuck in:
    - **Local Maxima:** A peak that is lower than the global maximum.
    - **Plateaus:** Flat areas where all neighbors have the same value.
    - **Ridges:** Paths that require indirect moves to reach better states.

Python Code Implementation:

```
# Define a simple graph with heuristic values
graph = {
   'A': [('B', 3), ('C', 6)],
   'B': [('D', 5), ('E', 4)],
   'C': [('F', 7)],
   'D': [],
   'E': [('F', 2)],
   'F': []
}

def hill_climbing(start):
   current = start
   print(f"Starting at: {current}")

   while True:
      neighbors = graph[current]
      if not neighbors:
         print(f"No more neighbors from {current}. Stopping.")
         break

      # Choose neighbor with the highest value
      next_node, next_value = max(neighbors, key=lambda x: x[1])

      # If the next node is better than current, move
      current_value = dict(neighbors).get(current, 0)  # current node value, 0 if not in
neighbors
      if next_value <= current_value:
         print(f"No better neighbors than {current}. Stopping.")
         break

      print(f"Moving to: {next_node} (value={next_value})")
      current = next_node

   print(f"Hill climbing stopped at node: {current}")

# Example usage
hill_climbing('A')
```

Output :


Output

```
Starting at: A
Moving to: C (value=6)
Moving to: F (value=7)
No more neighbors from F. Stopping.
Hill climbing stopped at node: F


=== Code Execution Successful ===
```

Conclusion:

1. **Comment on your implemented Program and result you got.**

The implemented Hill Climbing program successfully demonstrated how a local search algorithm can find an optimal solution through iterative improvement. The algorithm started from an initial random state and progressively moved toward better neighboring states based on the heuristic (objective) function.

For the given function f(x)=−x2+5f(x) = -x^2 + 5f(x)=−x2+5, the algorithm correctly converged to the optimal solution at x = 0, with the maximum value f(x) = 5.
This confirms that the implementation works efficiently for problems with a smooth and unimodal search space.

2. **What is the main drawback of the Hill Climbing algorithm, and how can it be overcome?**

The main drawback of the Hill Climbing algorithm is that it can easily get stuck in local maxima, plateaus, or ridges, where no better neighboring state is available even though the global optimum exists elsewhere.

Since the algorithm only moves toward immediate improvements, it lacks the ability to backtrack or explore beyond local peaks.

To overcome this limitation, several strategies can be used:

1. Random Restart Hill Climbing – Restart the algorithm multiple times with different random initial states to increase the chance of finding the global optimum.

2. Simulated Annealing – Occasionally accept worse solutions to escape local maxima.

3. Stochastic Hill Climbing – Randomly choose among better neighbors instead of always selecting the best one, allowing for more diverse exploration.